REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

AD-A243 602

REPORT DATE	3. REPORT TYPE AND DATES COVERED
	THESIS/DISSERTATION

4. TITLE AND SUBTITLE
Optimal Specialization and Allocation of Maintenance Manpower

5. FUNDING NUMBERS

6. AUTHOR(S)
Dennis C. Dietz, Major

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
AFIT Student Attending: Pennsylvania State University

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/CI/CIA- 91-016D

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFIT/CI
Wright-Patterson AFB OH 45433-6583

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release IAW 190-1
Distributed Unlimited
ERNEST A. HAYGOOD, Captain, USAF
Executive Officer

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

14. SUBJECT TERMS

15. NUMBER OF PAGES
109

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

16D

Abstract
Optimal Specialization and Allocation of Maintenance Manpower
Dennis C. Dietz, Major, USAF
Ph.D.; 1991
The Pennsylvania State University
109 pages

This thesis develops an analytical method for determining an optimal level of specialization and optimal task allocation for a maintenance manpower force. The method assumes that maintenance tasks are generated by a system of identical machines which experience random malfunctions and require periodic service. The impact of alternative manpower structures on system performance is evaluated using a queuing network model. Markov decision analysis is employed to determine an optimal assignment of manpower resources to pending tasks as the network status varies over time. A linear programming algorithm is derived to enable simultaneous optimization of specific assignment decisions and the overall manpower structure. The optimization method is developed and demonstrated through a simple example, but the dimensionality issues associated with larger system models are also addressed. The method is specifically applied to the problem of maximizing military aircraft sortie generation subject to a constraint on maintenance manpower expenditure.

The Pennsylvania State University

The Graduate School

Department of Industrial and Management Systems Engineering

OPTIMAL SPECIALIZATION AND ALLOCATION OF MAINTENANCE MANPOWER

A Thesis in

Industrial Engineering and Operations Research

by

Dennis C. Dietz

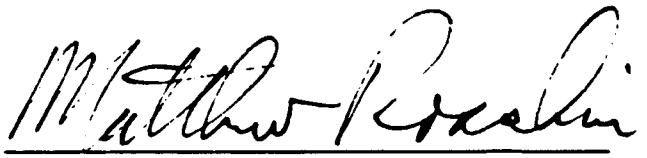
Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 1991

We approve the thesis of Dennis C. Dietz.

Date of Signature



6/3/91

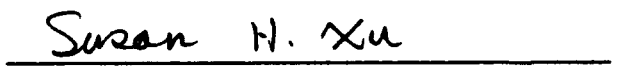
Matthew Rosenshine
Professor of Industrial Engineering
Thesis Adviser
Chair of Committee

6/3/91

Tom M. Cavalier
Associate Professor of Industrial Engineering

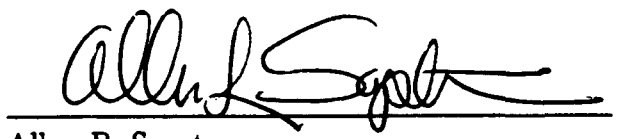
6/3/91

Brian J. Melloy
Assistant Professor of Industrial Engineering



6/3/91

Susan Hong Xu
Assistant Professor of Management Science



6/6/91

Allen B. Soyster
Professor of Industrial Engineering
Head of the Department of Industrial and
Management Systems Engineering

Abstract

This thesis develops an analytical method for determining an optimal level of specialization and optimal task allocation for a maintenance manpower force. The method assumes that maintenance tasks are generated by a system of identical machines which experience random malfunctions and require periodic service. The impact of alternative manpower structures on system performance is evaluated using a queuing network model. Markov decision analysis is employed to determine an optimal assignment of manpower resources to pending tasks as the network status varies over time. A linear programming algorithm is derived to enable simultaneous optimization of specific assignment decisions and the overall manpower structure. The optimization method is developed and demonstrated through a simple example, but the dimensionality issues associated with larger system models are also addressed. The method is specifically applied to the problem of maximizing military aircraft sortie generation subject to a constraint on maintenance manpower expenditure.

Contents

Figures

Tables

Symbols

A Set of task types which are always required between operational activities

a_m Number of maintenance personnel required for a type m task

B Set of group sizes for dependent machine operations

b Index for dependent machine group sizes ($b \in B$)

C Expenditure limit for maintenance manpower

c_y Unit cost of a type y maintenance specialist

D_{gi} Set of nondominated feasible decisions for manpower structure g and system state i

E_z Set of eligible tasks for network station z

G Total number of maintenance manpower structures

g Index for maintenance manpower structures ($g \in \{1, 2, \ldots, G\}$)

γ Index for transition activities ($\gamma \in \{0, 1, \ldots, M\}$)

H Total number of manpower specialization strategies

h Index for specialization strategies ($h \in \{1, 2, \ldots, H\}$)

I Total number of system states

i, j Indices for system states ($i, j \in \{1, 2, \ldots, I\}$)

k Index for manpower assignment decisions

L_y Training time for type y maintenance specialty

λ_m Malfunction rate for generation of type m tasks

M Total number of maintenance task types

m Index for maintenance task types ($m \in \{1, 2, \ldots, M\}$)

μ_0 Operational activity rate

μ_m Maintenance rate for a type m task

N Total number of machines in system

n_z Number of machines at network station z

P_z Set of pending tasks for network station z

p_{gik} Joint probability of employing manpower structure g, finding system in state i, and selecting assignment decision k

π_i Steady state probability that system is in state i

Q_y Set of qualified tasks for a type y maintenance specialist

q_z Routing probability from operational station to maintenance station z

R Aircraft sortie generation rate

r_{ij}^k Transition rate from state i to state j under assignment decision k

ρ^b Probability that an operational activity requires dependent machines of group size b

s_{ij} Number of maintenance personnel assigned to a transition task from system state i to state j

T_γ Time until completion of next type γ activity

t Time

τ Index for term of military service

U_τ Average value of annual pay and benefits for personnel in service term τ

V_y Reenlistment bonus for type y maintenance specialists

v_i Relative value of finding system in state i

w_τ Portion of maintenance workforce in service term τ

x_y Number of type y maintenance specialists employed in a manpower structure

Y Total number of maintenance specialty types

y Index for maintenance specialty types ($y \in \{1, 2, \ldots, Y\}$)

Z Total number of maintenance stations in system queuing network

z Index for network stations ($z \in \{0, 1, \ldots, Z\}$)

Φ_m Set of task types which must be completed before a type m task can be initiated

Chapter 1

Introduction

1.1 Motivation

The United States Air Force operates an inventory of over 3,000 tactical fighter aircraft at locations throughout North America, Europe, and the Pacific. These aircraft are extremely complex machines which require maintenance support from a variety of specialized personnel. Due to pressures for peacetime monetary efficiency, fighter units and support forces typically operate at fixed installations which can harbor large concentrations of aircraft. These centralized forces are becoming increasingly vulnerable to qualitative improvements in the aerial bombardment and surface-to-surface missile capabilities of potential wartime adversaries. Consequently, Air Force leadership has endorsed a deployment strategy which "calls for decentralized, small-unit autonomy with the mobility and flexibility to survive and sustain dispersed combat operations" [38, p. 2]. Planners are now considering unit deployments involving very small numbers of aircraft. Unfortunately, such decentralization could substantially increase maintenance manpower requirements. One study has shown that dispersing aircraft to small unit bases could increase needed manpower by two-thirds just to ensure the availability of minimum crew sizes for all specialized tasks [4]. Even more manpower would be needed to limit queuing delays to a level where current standards of operational effectiveness could be maintained.

The apparent need for more manpower competes with the constraints im-

posed by a decreasing budget and shrinking supply of enlistment-aged citizens. An alternative approach under consideration is a restructuring of maintenance specialties so that required tasks can be accomplished by fewer personnel having a wider range of skills. The higher costs of recruiting, training, and retaining maintenance "generalists" may be more than offset by the efficient manpower utilization that could be preserved under dispersed operations. This possibility motivates an important question: what specialization strategy will maximize operational effectiveness for a given level of manpower expenditure?

1.2 Problem Statement

The general problem of optimal specialization for maintenance manpower could apply to myriad military and industrial concerns. Many enterprises depend on the steady operation or availability of complex machinery. For example, a transportation entity (airline, trucking firm, overnight delivery service, etc.) cannot function competitively without a well-maintained fleet of vehicles. A manufacturing facility may rely critically on sophisticated production equipment. Timely maintenance of these machines requires skills which are expensive to develop, and the personnel possessing these skills are valuable resources. Nevertheless, the division of labor among maintenance personnel can often reflect traditional organizational boundaries rather than efficient allocation with respect to the overall objectives of the parent enterprise. This thesis presents a general method for determining an optimal *specialization strategy*, that is, for optimizing the number of maintenance specialties and the allocation of tasks to specialties. The optimization method explicitly recognizes the economic tradeoff between the lower per-person costs and the lower average utilizations resulting from specialization.

As indicated in Figure 1.1, determination of an optimal specialization strat-

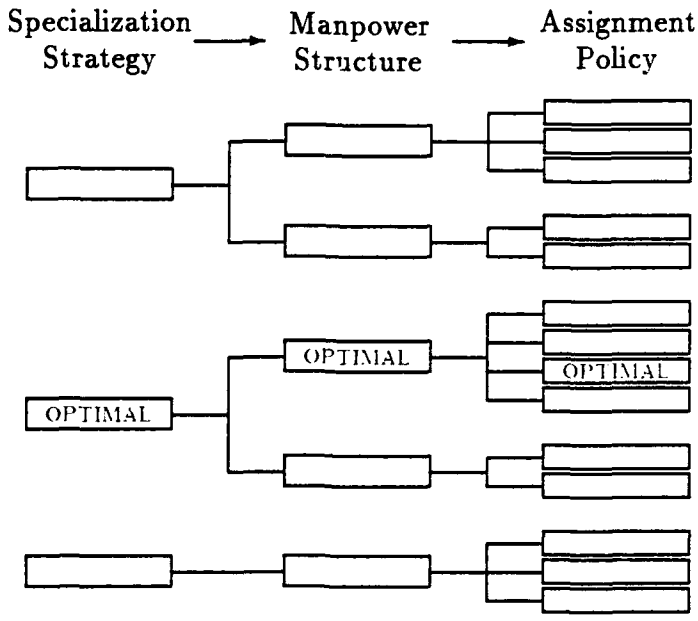


Figure 1.1: Components of the Manpower Problem

egy requires suboptimization at two levels. First, an optimal strategy cannot be identified without determining an optimal *manpower structure*, which specifies exact quantities for each type of specialist in the manpower force. Further, to find an optimal structure, it is necessary to determine an optimal *assignment policy*. The assignment policy defines comprehensive decision rules for dispatching manpower to specific pending tasks as the enterprise operates over time. The number of possible manpower structures and assignment policies can be quite large for a particular application, so determining an optimal specialization strategy by exhaustive enumeration may be computationally intractable. Enumerative evaluation of all manpower structures and assignment policies can be avoided through the analytical method developed in this thesis.

1.3 Assumptions

The approach pursued in developing the optimization method assumes that the modeled enterprise has the general characteristics listed below. Relaxation of some of these assumptions is considered as the model development progresses.

1. The enterprise employs a fixed inventory of N machines with identical potential maintenance requirements. Any machine not requiring maintenance is immediately utilized in operations.

2. Operations are conducted continuously, so the goal of the enterprise is to maximize steady-state performance. This is achieved by maximizing the expected number of machines operating at a random point in time.

3. A finite set of M tasks describes all possible types of machine maintenance. Each task $m \in \{1, 2, \ldots, M\}$ is characterized by a required number of personnel a_m and a known average completion rate μ_m. The time required to perform each task is represented by an exponentially distributed random variable [33, pp. 201–209].

4. A required maintenance task can be generated by a corresponding type of random malfunction, or by completion of a time-limited operational activity (e.g., refueling after each aircraft sortie). The operating time between random malfunctions of any type m is represented by an exponentially distributed random variable with rate λ_m for each machine. Each type of random malfunction is independent from all other types, and all machines operate independently.

5. The duration of each operational activity is represented by an exponentially distributed random variable with rate μ_0. Maintenance tasks generated due to

random malfunctions can be deferred until completion of the next operational activity, so a machine can reach a condition where several types of tasks are required. In this case, the required tasks can be performed sequentially or in parallel, depending on a specified compatibility of the task types.

6. Each type of maintenance task must be allocated to a particular type of maintenance specialty. The number of specialties in a particular specialization strategy can range from one (one specialty for all tasks) to M (a separate specialty for each task). Thus, the number of specialties Y to be considered across all possible strategies cannot exceed an upper bound of $\sum_{m=1}^{M}\binom{M}{m} = 2^M - 1$.

7. Continuous manning of any maintenance specialty $y \in \{1, 2, \ldots, Y\}$ incurs a known cost of c_y units per specialist. The enterprise is constrained to an upper bound of C units for total maintenance manpower expenditure. Thus, any feasible manpower structure must satisfy the constraint $\sum_{y=1}^{Y} c_y x_y \leq C$ where x_y is the number of type y specialists employed under the structure.

These assumptions provide a framework for analyzing the interacting effects of machine characteristics and manpower availability on overall system performance. At any point in time, an individual machine may be operating, or it may require a finite number of maintenance tasks. The state of the overall system can be described by listing the number of machines in each possible condition. If the number of machines requiring a particular maintenance task exceeds the allocated number of specialists, the excess machines will queue for an available specialist and potential operating time will be lost. Thus, each maintenance condition could correspond to a station in a "queuing network," with the stations connected by the possible movement of a machine from one condition to another. Since operating times and maintenance times are modeled with exponential distributions,

the movement of machines through the network can be described as a continuous-time Markov process [37, pp. 39–45]. Several analytical methods can be applied to evaluate a system conforming to this general model.

1.4 Literature Review

The published literature generally associated with the disciplines of industrial engineering, management science, and operations research makes scarce reference to the specific issue of optimal manpower specialization. However, the general topic of resource utilization in a queuing environment is well developed, and many theoretical contributions can be applied to the manpower problem. In addition, several government studies have quantitatively addressed some manpower issues associated with complex military systems. Many types of machines with defense applications (particularly aircraft) tend to have a high ratio of maintenance man-hours to operating hours, and the resources required to perform maintenance are expensive.

1.4.1 Queuing Network Analysis

The high requirements and costs of aircraft maintenance motivated the first published investigations of queuing network behavior. In 1954, the concept of a cyclic queue was introduced by J. Taylor and R. R. P. Jackson as a model for relating maintenance rates and spare engine supply to the availability of a fleet of aircraft [36]. Taylor and Jackson considered the movement of a finite supply of customers (aircraft engines) through a series of service stations with infinite queue capacities and exponentially distributed service times. In the same journal, R. R. P. Jackson published another article which addressed a similar problem with an infinite arrival population [19]. Jackson theorized that the output distribution of a

service station was identical to the input distribution. This result, proved by Burke in 1956 [7], permitted independent treatment of each queue in a series. In 1957, J. R. Jackson published *Networks of Waiting Lines*, which formalized the earlier results of R. R. P. Jackson and extended them to include systems of queues in which transitions of customers from one queue to another at service completion are random events [20]. Such systems are now often referred to as "Jackson networks."

In the original Jackson network model, customers arrive from an infinite external source according to a Poisson process. Each service station contains one or more parallel servers, and all service times are exponentially distributed. Any state of a system with $Z + 1$ stations can be represented by a vector $\vec{n} = (n_0, n_1, \ldots, n_Z)$ where $n_z \geq 0$ is the number of customers at each station $z \in \{0, 1, \ldots, Z\}$. The equilibrium probability that the network is in state $\vec{n}$ is denoted by $P(\vec{n}) = P(n_0, n_1, \ldots, n_Z)$, and the marginal probability of finding n_z customers at a station z is denoted by $p_z(n_z)$. Jackson proved that the equilibrium probability of a given state $\vec{n}$ can be factored into the product of each of the marginal distributions; that is,

$$P(n_0, n_1, \ldots, n_Z) = \prod_{z=0}^{Z} p_z(n_z) \tag{1.1}$$

This "product form" relationship, coupled with Burke's earlier result, greatly simplified the analysis of many queuing systems.

In 1967, Gordon and Newell extended the product form result to obtain the equilibrium state probabilities for closed Markovian networks [14]. They studied a system with a finite number of customers N cycling through a network of stations with multiple parallel servers. The state of the system is again described by a vector $\vec{n} = (n_0, n_1, \ldots, n_Z)$, but now an additional constraint is imposed; that is,

$\sum_{z=0}^{Z} n_z = N$. Gordon and Newell proved that the equilibrium probability of state $\vec{n}$ also satisfies a product form

$$P(n_0, n_1, \ldots, n_Z) = \frac{1}{\Gamma(N)} \prod_{z=0}^{Z} f_z(n_z) \tag{1.2}$$

where each function f_z depends only on the characteristics of the zth station and $\Gamma(N)$ is a normalization constant chosen to make all the feasible state probabilities sum to one. Although the product form developed by Gordon and Newell was easily expressed, direct calculation of the state probabilities was often computationally expensive and inaccurate because of the large number of operations required to evaluate the normalization constant. In 1973, J. P. Buzen developed an efficient algorithm for evaluating $\Gamma(N)$ [8]. Buzen also studied methods for obtaining the performance measures of a queuing system as a simple function of the model parameters and the normalization constant.

Certain types of queuing networks do not conform to the product form model. In 1978, P. J. Denning surveyed an "operational approach" to queuing system analysis and thoroughly studied the conditions required for solution of queuing network models. The first condition needed for determination of equilibrium state probabilities is "flow balancing," meaning that the transition rates in and out of each system state must be balanced. This condition permits construction of a set of global balance equations (one equation for each state) which can be solved simultaneously by traditional methods. Unfortunately, even modest-sized networks can generate a large number of simultaneous equations. The product form approach circumvents this problem, but requires that two additional conditions be met:

1. Multiple customer arrivals and departures are not observed, so that the rate at which the system enters or leaves a state depends only on the rate of customer

flow between stations ("one-step behavior").

2. The flow rate out of a station depends only on the station's queue length, and not on how customers are distributed elsewhere in the system ("homogeneity").

Under these circumstances, the global balance equations decompose into a set of local balance equations and the product form result can be applied. However, not all queuing systems satisfy the required conditions. For example, the homogeneity condition is violated by systems which involve some sort of "blocking," where an event at one station can prevent another station from serving customers. This can occur if resource conflicts exist, such as when queues must share servers. Unfortunately, this is precisely the situation that arises when several maintenance tasks require one type of resource and total task requirements exceed the number of available resources. Thus, the manpower specialization problem requires solution techniques which do not rely on product form methods.

Researchers concede a lack of progress in obtaining exact solutions for large queuing networks that do not have product form state probabilities [29]. Two principal approximation methods for attacking these networks have emerged: diffusion and decomposition. The diffusion approximation attempts to reduce the complexity of the global balance equations by treating individual queue lengths n_z as continuous random variables [21]. This technique can be particularly useful for handling nonexponential service time distributions and for finding transient solutions [22]. Unfortunately, diffusion methods can be difficult to implement and are based on an assumption of "heavy traffic" which would always favor highly specialized manpower structures. Thus, approximation methods based on diffusion are less applicable to the manpower specialization problem than approximation methods

based on decomposition.

The decomposition approach was introduced in 1975 by P. J. Courtois as a tool for performance evaluation of complex computer systems [11]. An example of this approach might consist of analyzing a subsystem in isolation and then replacing the subsystem with a single composite server which imitates the behavior of the originally isolated subsystem. Courtois reported that such an approximation is accurate if the rate of interaction within the subsystem is substantially higher than the rate of interaction between the subsystem and the rest of the overall network. The decomposition approach was further developed by Chandy, Herzog, and Woo, who applied Norton's theorem from electrical circuit theory to queuing networks which obey local balance [9]. Chandy et al. demonstrated that, given a network with $Z + 1$ stations, it is possible to replace Z of the stations with a single composite server having load-dependent service rates. They also presented an approximate procedure for modeling this "complementary queue" in a network which does not obey local balance. These techniques permit construction of approximate models of otherwise intractable systems by defining stations which are "flow equivalent" to major portions of an original network. This notion of flow equivalence has significant applicability to the problem of determining an optimal manpower specialization strategy for a large system.

1.4.2 Military Manpower Analysis

The high maintenance requirements of complex military systems suggest that maintenance manpower would be a major consideration in system design. However, quantitative methods have not always been applied to thoroughly address manpower issues. In 1960, a research memorandum produced by the RAND Corporation for the U. S. Air Force reported:

> Manpower planning, until recently, has played a rather passive role in the research and development of future Air Force weapon and support systems. Plans for the best utilization of people in new systems have been developed late in the R&D cycle and usually after decisions have been made about hardware and basic operational characteristics. [16, p. 1]

The RAND memorandum described a general approach for analyzing manpower requirements, but did not provide specific quantitative techniques. In the years since this early report, manpower issues have been studied extensively, but almost exclusively through models which employ discrete event simulation of logistic systems [30]. One notable exception is the DYNA-METRIC model, which is an analytical tool used primarily to measure the impact of spare parts management on aircraft sortie generation [17, 34]. In current manpower analysis, the tool most frequently used is a widely accepted computer simulation known as the Logistics Composite Model (LCOM).

Simulation models can be constructed to capture an arbitrary level of detail, but the price for this advantage includes uncertain accuracy, substantial expense, and a lack of analytical insight into causal relationships. Simulation is particularly poorly suited for optimization problems, since the random nature of results hinders iterative convergence on an optimal combination of many input variables. Mathematical queuing models have occasionally been employed in manpower studies, but usually in analyzing particular system components rather than the interactions between all maintenance requirements and available manpower. The manpower specialization problem addressed in this thesis involves a broader context than has thus far been addressed with mathematical methods.

In 1964, H. R. Barton and others developed an early analytical method for investigating the logistical requirements of alternative system designs [1, 32]. The method employed queuing tables to analyze tradeoffs between manpower levels,

spare parts inventory, and machine downtime. Machine components were considered separately for specified levels of service demand that were externally determined. This general approach typified subsequent applications of queuing theory to military logistics problems, including the DYNA-METRIC model and its predecessors [17, p. 11].

In 1975, J. R. Phelan employed simple queuing network analysis to predict the impact of maintenance manpower levels on aircraft operational effectiveness. Different types of maintenance requirements were modeled in separate cycles, and interactions were explicitly ignored [31, p. 23]. Interestingly, the computational methods used in this analysis relied on global balance, even though product form solution techniques could have been applied. The effort also presented some justification for considering manpower issues separately from other dimensions of the overall logistics system (spare parts, etc.).

Beginning in the mid 1980s, the specific problem of optimal task allocation for aircraft maintenance specialties became the subject of keen interest. This interest was partially motivated by increased equipment sophistication and maintenance complexity, which raised the aptitudes and skills required of maintenance personnel. Competing demands for the same aptitudes and skills in other occupations (military and industrial), coupled with projected demographic changes, presented serious manpower challenges. These challenges were amplified by an institutional goal to improve aircraft survivability through dispersal, an operational concept that would stretch maintenance resources even further.

In 1983, M. Berman and C. Batten used the TSAR simulation model (Theater Simulation of Airbase Resources) to estimate the number of aircraft sorties that could be generated under differing degrees of dispersal [4]. Variables considered in the analysis included manpower, spare parts, and aircraft reliability and

maintainability characteristics. While Berman and Batten did not explicitly consider maintenance costs, they concluded that specialty consolidation could improve operational performance or reduce manpower requirements.

Simulation analysis was also employed by C. H. Shipman in 1985 to evaluate the potential manpower savings realizable through specialty consolidation. He employed the Logistics Composite Model to simulate ground attack fighter operations. Shipman found that a dispersed unit of eighteen aircraft could be maintained by 27% fewer technicians through "minor" specialty consolidation (combining two or three specialties into one), and by 37% fewer technicians through "major" specialty consolidation (combining six specialties into one). Shipman recognized the existence of "tradeoffs between the manpower savings of combining specialties and the additional skill/training requirements these combinations generate" [35, p. 16]. He recommended that these tradeoffs be studied to determine an optimal level of consolidation.

In 1986, G. A. Gotz and R. E. Stanton considered the impact of maintenance specialty cross-training on operational effectiveness [15]. They simulated a simple aircraft operation that involved two types of repairable components and two maintenance specialties. Gotz and Stanton concluded that cross-training is particularly important if wartime failure rates and repair rates differ significantly from pre-war expectations. They also pointed out that, when maintenance specialists with multiple skills are introduced, decision rules must also be implemented to specify which malfunctioning components will be repaired first.

The Air Force recently engaged in a major initiative to broadly address maintenance manpower issues. The initiative, called "Rivet Workforce," succeeded in generating some institutional momentum toward specialty consolidation. The resulting need for an analytical framework to evaluate particular consolidation strate-

gies motivated a recent study effort called SUMMA, or Small Unit Maintenance Manpower Analysis [24, 28, 38]. A major product of the SUMMA project is a computerized decision aid to assist in the derivation of optimal task allocations. The computer model employs a practical analytical method which is based on assumptions that all sortie durations and maintenance times are deterministic, all aircraft fly sorties and receive maintenance in "batches," and all aircraft maintenance tasks are performed in series. The task allocations and manpower estimates derived through the analytical method are refined through iterative simulation experiments. While the complexity of the overall maintenance system guarantees a significant role for simulation modeling, substantial insight can be gained through additional mathematical treatment of the problem. The system structure suggests a meaningful role for queuing network analysis.

Chapter 2

Model Development

Development of a manpower specialization model using queuing network analysis is perhaps best illustrated through a simple example. In this chapter, global balance is used to obtain an optimal specialization strategy for a small commercial enterprise called "Mike's Flying Club."

2.1 Problem Formulation

Mike operates an around-the-clock flying service at a municipal airport. His rented facility will support only two aircraft ($N = 2$), and demand for flying sorties is high enough that both machines can be employed whenever they are available. Mike attributes this high demand to his reasonable fees, so he is very interested in controlling his expenses. He is particularly concerned about the high costs of good aircraft mechanics and is determined to keep his maintenance payroll below \$100 per hour ($C = 100$).

In the course of day-to-day operations, Mike's airplanes can require any of three different types of maintenance tasks:

1. Routine "turn-around" maintenance, such as refueling and routine inspection, which is always required between sorties. This task is completed at an hourly rate $\mu_1 = 1.0$ by a single mechanic ($a_1 = 1$).

2. Unplanned "airframe" maintenance, which is required when an aircraft returns from a sortie with a reported malfunction not related to the aircraft engine.

This task is accomplished at rate $\mu_2 = .25$, and also requires one mechanic ($a_2 = 1$).

3. Unplanned "engine" maintenance, which is required when an aircraft returns with reported engine problems. The service rate for this task is $\mu_3 = .5$, and two mechanics are required ($a_3 = 2$).

Occasionally, a returning aircraft will require both airframe and engine maintenance. In this case, the two types of repair can be initiated simultaneously, provided sufficient mechanics are available. Turn-around maintenance is never initiated until an aircraft completes all unplanned airframe and engine maintenance. These relationships are illustrated in the network representation shown in Figure 2.1. Each station in the network can be characterized by a set of pending tasks P_z and a set of eligible tasks E_z. Pending tasks are those which must be accomplished before a machine at the station will become operational. Eligible tasks are those which can be initiated immediately if qualified manpower is available.

The number of states realizable by the complete system is determined by the number of ways in which $N = 2$ aircraft can occupy $Z + 1 = 5$ stations. This is a standard occupancy problem [13, p. 36], so the total number of states I can be calculated as $\binom{N+Z}{Z} = \binom{6}{4} = 15$. The system will transition to a new state whenever an aircraft completes an activity (sortie or maintenance task) and moves to a new station. Since all sortie durations and maintenance times are modeled with exponential distributions, the system is Markovian and exhibits one-step behavior. Each feasible transition from one state i to another state j can be characterized by a losing station $z^-(ij)$ and a gaining station $z^+(ij)$. This information is portrayed in Table 2.1, along with an index $\gamma(ij)$ which identifies the activity which an aircraft completes at transition. A value of $\gamma(ij) = 0$ implies completion of an aircraft

0: aircraft operating

1: turn-around maintenance
$P_1 = \{1\}, E_1 = \{1\}$

2: airframe maintenance
$P_2 = \{1,2\}, E_2 = \{2\}$

3: engine maintenance
$P_3 = \{1,3\}, E_3 = \{3\}$

4: airframe and engine maintenance
$P_4 = \{1,2,3\}, E_4 = \{2,3\}$

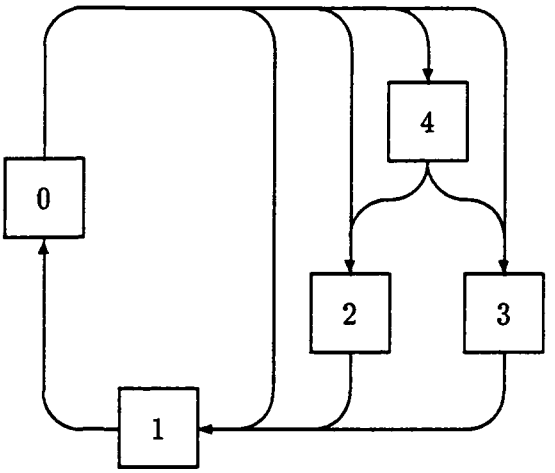


Figure 2.1: Possible Conditions for Individual Aircraft

sortie, while a value of $\gamma(ij) \in \{1,2,3\}$ implies completion of a corresponding type of maintenance task.

The transition rates between system states will depend on the sortie rate, maintenance rates, and the number of personnel available to perform the various types of maintenance. It is also necessary to translate aircraft malfunction rates into network routing probabilities for aircraft as they complete sorties. Aircraft maintenance records for Mike's operation reveal average airframe and engine malfunction rates of $\lambda_2 = .2$ and $\lambda_3 = .25$ respectively. Flight log-books indicate that an average sortie has a duration of 2.0 hours, yielding an average sortie completion rate of $\mu_0 = .5$. For any aircraft beginning a sortie, let T_0 be the sortie completion time, let T_2 be the time until the next airframe malfunction, and let T_3 be the time until the next engine malfunction. Routing probabilities q_z for $z \in \{1,2,3,4\}$ can

Table 2.1: System States and Transitions (z^-, z^+)

$\vec{n}(i)$ 01234	i	j														
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
20000	1		0 1 0	0 2 0	0 3 0	0 4 0										
11000	2	1 0 1					0 1 0	0 2 0	0 3 0	0 4 0						
10100	3		2 1 2					0 1 0			0 2 0	0 3 0	0 4 0			
10010	4		3 1 3						0 1 0			0 2 0		0 3 0	0 4 0	
10001	5			4 2 3	4 3 2					0 1 0			0 2 0		0 3 0	0 4 0
02000	6		1 0 1													
01100	7			1 0 1			2 1 2									
01010	8				1 0 1		3 1 3									
01001	9					1 0 1		4 2 3	4 3 2							
00200	10							2 1 2								
00110	11							3 1 3	2 1 2							
00101	12									2 1 2	4 2 3	4 3 2				
00020	13								3 1 3							
00011	14									3 1 3		4 2 3		4 3 2		
00002	15												4 2 3		4 3 2	

be computed as follows:

$$\begin{aligned} q_1 &= P(T_0 < T_2, T_0 < T_3) \\ &= P\{T_0 < \min(T_2, T_3)\} \\ &= \frac{\mu_0}{\mu_0 + \lambda_2 + \lambda_3} = \frac{.5}{.5 + .25 + .2} = .5263 \end{aligned}$$

$$\begin{aligned} q_2 &= P(T_0 \geq T_2, T_0 < T_3) \\ &= P(T_0 < T_3) - P(T_0 < T_2, T_0 < T_3) \\ &= \frac{\mu_0}{\mu_0 + \lambda_3} - q_1 = \frac{.5}{.5 + .25} - .5263 = .1404 \end{aligned}$$

$$\begin{aligned} q_3 &= P(T_0 < T_2, T_0 \geq T_3) \\ &= P(T_0 < T_2) - P(T_0 < T_2, T_0 < T_3) \\ &= \frac{\mu_0}{\mu_0 + \lambda_2} - q_1 = \frac{.5}{.5 + .2} - .5263 = .1880 \end{aligned}$$

$$\begin{aligned} q_4 &= P(T_0 \geq T_2, T_0 \geq T_3) \\ &= 1 - P(T_0 < T_2, T_0 < T_3) - P(T_0 \geq T_2, T_0 < T_3) - P(T_0 < T_2, T_0 \geq T_3) \\ &= 1 - q_1 - q_2 - q_3 = 1 - .5263 - .1404 - .1880 = .1454 \end{aligned}$$

These computations can be generalized for an arbitrary network with any number of stations. First, let A be the set of tasks which are always required between operational activities. Clearly, any station whose pending tasks do not include all tasks in A can not be entered directly from station 0. Thus, $q_z = 0 \; \forall \; z : A \not\subseteq P_z$.

For all other stations,

$$\begin{aligned} q_z &= P(T_0 < T_m \ \forall\, m \notin P_z \cup A) - \sum_{\zeta: P_\zeta \subset P_z} q_\zeta \\ &= \frac{\mu_0}{\mu_0 + \sum_{m \notin P_z \cup A} \lambda_m} - \sum_{\zeta: P_\zeta \subset P_z} q_\zeta \end{aligned} \tag{2.1}$$

The routing probabilities are well suited for sequential computation if the maintenance stations are ordered by increasing numbers of tasks in their pending task sets.

Once the routing probabilities are computed, global balance equations can be constructed to compute the equilibrium state probabilities π_i resulting from a particular manpower assignment policy. To facilitate a general representation of an assignment policy, it is useful to define a variable s_{ij} as the number of mechanics assigned to perform each transition task $\gamma(ij) \in \{1, 2, \ldots, M\}$. The maximum value of each s_{ij} is limited by the number of machines requiring maintenance; i.e., $s_{ij} \leq a_{\gamma(ij)} n_{z^-(ij)} \ \forall\ i, j \in \{1, 2, \ldots, I\}$. The values are also constrained by manpower availability; i.e., $\sum_{j:\gamma(ij) \in Q_y} s_{ij} \leq x_y \ \forall\, i \in \{1, 2, \ldots, I\}, y \in \{1, 2, \ldots, Y\}$. Since the system is Markovian, the rate of a transition task completion can clearly be represented as $[s_{ij}/a_{\gamma(ij)}]\mu_{\gamma(ij)}$. It is also clear that this expression must reduce to $\mu_{\gamma(ij)}$ when task $\gamma(ij)$ is the only one pending in the system, since there would be no potential for a resource conflict. Using these facts and the information presented in Table 2.1, global balance equations for each state of the example can be constructed as follows (rate out=rate in):

$$\pi_1 2\mu_0 = \pi_2 \mu_1 \tag{2.2}$$

$$\pi_2(\mu_1 + \mu_0) = \pi_1 2 q_1 \mu_0 + \pi_3 \mu_2 + \pi_4 \mu_3 + \pi_6 s_{62} \mu_1 \tag{2.3}$$

$$\pi_3(\mu_2+\mu_0)=\pi_1 2q_2\mu_0+\pi_5[\frac{s_{53}}{2}]\mu_3+\pi_7 s_{73}\mu_1 \tag{2.4}$$

$$\pi_4(\mu_3+\mu_0)=\pi_1 2q_3\mu_0+\pi_5 s_{54}\mu_2+\pi_7 s_{84}\mu_1 \tag{2.5}$$

$$\pi_5\{[\frac{s_{53}}{2}]\mu_3+s_{54}\mu_2+\mu_0\}=\pi_1 2q_4\mu_0+\pi_9 s_{95}\mu_1 \tag{2.6}$$

$$\pi_6 s_{62}\mu_1=\pi_2 q_1\mu_0+\pi_7 s_{76}\mu_2+\pi_8[\frac{s_{86}}{2}]\mu_3 \tag{2.7}$$

$$\pi_7\{s_{73}\mu_1+s_{76}\mu_2\}=\pi_2 q_2\mu_0+\pi_3 q_1\mu_0+\pi_9[\frac{s_{97}}{2}]\mu_3+\pi_{10}s_{10\,7}\mu_2+\pi_{11}[\frac{s_{11\,7}}{2}]\mu_3 \tag{2.8}$$

$$\pi_8\{s_{83}\mu_1+[\frac{s_{86}}{2}]\mu_3\}=\pi_2 q_3\mu_0+\pi_4 q_1\mu_0+\pi_9 s_{98}\mu_2+\pi_{11}s_{11\,8}\mu_2+\pi_{13}[\frac{s_{13\,8}}{2}]\mu_3 \tag{2.9}$$

$$\pi_9\{s_{95}\mu_1+[\frac{s_{97}}{2}]\mu_3+s_{98}\mu_2\}=\pi_2 q_4\mu_0+\pi_5 q_1\mu_0+\pi_{12}s_{12\,9}\mu_2+\pi_{14}[\frac{s_{14\,9}}{2}]\mu_3 \tag{2.10}$$

$$\pi_{10}s_{10\,7}\mu_2=\pi_3 q_2\mu_0+\pi_{12}[\frac{s_{12\,10}}{2}]\mu_3 \tag{2.11}$$

$$\pi_{11}\{[\frac{s_{11\,7}}{2}]\mu_3+s_{11\,8}\mu_2\}=\pi_3 q_3\mu_0+\pi_4 q_2\mu_0+\pi_{12}s_{12\,11}\mu_2+\pi_{14}[\frac{s_{14\,11}}{2}]\mu_3 \tag{2.12}$$

$$\pi_{12}\{s_{12\,9}\mu_2+[\frac{s_{12\,10}}{2}]\mu_3+s_{12\,11}\mu_2\}=\pi_3 q_4\mu_0+\pi_5 q_2\mu_0+\pi_{15}[\frac{s_{15\,12}}{2}]\mu_3 \tag{2.13}$$

$$\pi_{13}[\frac{s_{13\,8}}{2}]\mu_3=\pi_4 q_3\mu_0+\pi_{14}s_{14\,13}\mu_2 \tag{2.14}$$

$$\pi_{14}\{[\frac{s_{14\,9}}{2}]\mu_3+[\frac{s_{14\,11}}{2}]\mu_3+s_{14\,13}\mu_2\}=\pi_4 q_4\mu_0+\pi_5 q_3\mu_0+\pi_{15}s_{15\,14}\mu_2 \tag{2.15}$$

$$\pi_{15}\{[\frac{s_{15\,12}}{2}]\mu_3+s_{15\,14}\mu_2\}=\pi_5 q_4\mu_0 \tag{2.16}$$

Since flow must be balanced for each state, any one of the above equations will be redundant. A solution for the state probabilities can be realized by arbitrarily deleting Equation 2.16 and replacing it with the normalizing constraint

$$\sum_{i=1}^{15}\pi_i=1 \tag{2.17}$$

Once the π_i are obtained by simultaneous solution of the balance equations, the

expected number of aircraft operating at a random point in time can be calculated as

$$E(n_0) = \sum_{i=1}^{15} n_0^{(i)} \pi_i = 2\pi_1 + \pi_2 + \pi_3 + \pi_4 + \pi_5 \tag{2.18}$$

Mike's expected sortie generation rate (sorties per aircraft per day) can be computed as

$$R = 24\mu_0 \frac{E(n_0)}{N} = (24)(.5)\frac{E(n_0)}{2} = 6E(n_0) \tag{2.19}$$

The sortie generation rate is a standard measure of effectiveness for aircraft operations.

It should be emphasized that the balance equations require specific input values s_{ij} for the number of mechanics designated to perform specific tasks at specific stations in each system state. Twenty-two of these values appear in Equations 2.2 through 2.16, and each must be treated as a variable in optimizing system performance. This is true because system performance will be determined not only by a prespecified manpower structure, but also by Mike's decisions on how the manpower will be utilized when each system state is encountered. For example, suppose Mike currently employs three types of mechanics, specializing in each of the three different types of maintenance tasks (maximum specialization). Specifically, suppose that Mike's payroll includes two turn-around mechanics ($x_1 = 2$), one airframe mechanic ($x_2 = 1$), and two engine mechanics ($x_3 = 2$). When the system is in state 12, so that $\vec{n} = (0, 0, 1, 0, 1)$, the single airframe mechanic could be assigned to perform an airframe task on the aircraft in condition 2 ($s_{12\,9} = 1, s_{12\,11} = 0$) or the aircraft in condition 4 ($s_{12\,9} = 0, s_{12\,11} = 1$). A similar conflict exists for engine mechanics in state 14, resulting in a total of $2 \times 2 = 4$ assignment policies. The potential for this kind of resource conflict increases if specialization is reduced so that a mechanic can perform more than one type of task. Thus, an optimization

method must consider not only an overall manpower structure, but also a comprehensive policy for assigning manpower to specific tasks when resource conflicts exist.

2.2 Optimal Specialization Strategy

Currently, Mike employs shifts of turn-around, airframe, and engine mechanics at hourly wages of \$10, \$20, and \$25, respectively. He has observed that mechanics are sometimes idle because their special skills don't always correspond with existing maintenance requirements. Mike knows he could improve manpower utilization by hiring mechanics who are qualified in multiple types of maintenance, but he would have to pay significantly higher wages. The hourly wage for each available type of mechanic $y \in \{1, 2, \ldots, 5\}$ is shown in Table 2.2. Also shown is the set of tasks Q_y for which each type of mechanic would be qualified.

Table 2.2: Costs and Skills of Available Manpower

y	1	2	3	4	5
c_y	\$10	\$20	\$25	\$30	\$33
Q_y	{1}	{2}	{3}	{2,3}	{1,2,3}

Mike wants to determine the maintenance manpower structure (and hence, the specialization strategy) that will maximize his sortie generation rate without exceeding his constrained level of manpower expenditure. A particular manpower structure can be represented by a vector $\vec{x} = (x_1, x_2, \ldots, x_5)$, where each x_y is the number of type y mechanics employed. An acceptable structure will satisfy the following conditions:

1. Total hourly manpower expenditure will not exceed the fixed budget; i.e., $\sum_{y=1}^{5} c_y x_y \leq 100$.

2. Each task type will be assigned to only one type of mechanic; i.e., if $Q_y \cap Q_{y'} \neq \emptyset$, then $x_y x_{y'} = 0 \; \forall \; y, y' \in \{1, 2, \ldots, 5\}, \; y \neq y'$.

3. Sufficient mechanics will be qualified to perform each type of task; i.e., $x_y \geq \max_{m \in Q_y}(a_m) \; \forall \; y \in \{1, 2, \ldots, 5\}$.

4. Mechanics of each type will have a potential for simultaneous utilization; i.e., $x_y \leq \max_{z \in \{1.2,3.4\}}(\sum_{m \in E_z \cap Q_y} a_m N) \; \forall \; y \in \{1, 2, \ldots, 5\}$.

Table 2.3 lists all manpower structures which satisfy the above constraints and require a total expenditure as close to \$100 as possible without exceeding this limit. Let $G = 5$ be the total number of candidate manpower structures and $H = 3$ be the total number of specialization strategies. For each structure $g \in \{1, 2, \ldots, G\}$ and corresponding strategy $h \in \{1, 2, \ldots, H\}$, the table lists the number of policies generated by all combinations of manpower assignment decisions. To find an optimal structure by exhaustive enumeration, it would be necessary to solve 4+2+24+12+36=78 sets of I simultaneous equations.

Fortunately, enumeration can be avoided by employing a continuous-time Markov decision model [18, pp. 92–114]. Development of the model begins with the generation of a list of manpower assignment decisions which are possible for each state across an unrestricted range of manpower structures. Table 2.4 displays all of the possible decisions k for the "Mike's Flying Club" example. For each specific structure g and system state i, a subset D_{gi} of nondominated feasible decisions can be identified. A nondominated feasible decision is one which does not permit any avoidable idleness of the resources available within a particular manpower structure.

Table 2.3: Manpower Structure Alternatives

g	h_g	x_1	x_2	x_3	x_4	x_5	Policies	Iterations	$E(n_0)$	R
1	1	2	1	2	0	0	4	1	.8080	4.848
2	1	1	2	2	0	0	2	1	.8159	4.895
3	2	2	0	0	2	0	24	2	.7900	4.740
4	2	1	0	0	3	0	12	2	.8103	4.862
5	3	0	0	0	0	3	36	2	.8409	5.045

For example, consider structure 5, which provides three mechanics qualified in all types of maintenance. Suppose the system is in state 15, so that two aircraft require airframe and engine maintenance. The set $D_{5\,15}$ contains two nondominated feasible decisions. One option would be to assign all the mechanics to one aircraft, with two mechanics performing the engine task and one mechanic performing the airframe task ($s_{15\,12} = 2, s_{15\,14} = 1$). Alternatively, two mechanics could be assigned to perform the airframe tasks on both aircraft, with one mechanic unavoidably idle ($s_{15\,12} = 0, s_{15\,14} = 2$). As indicated in Table 2.4, these options correspond to decisions 5 and 7, so $D_{5\,15} = \{5, 7\}$. The nondominated feasible decisions for every manpower structure and system state are displayed in Table 2.5.

Once all nondominated feasible decisions are identified, an efficient Markov decision algorithm can be employed. It can first be noted that the underlying Markov process is completely ergodic, meaning that the steady-state probabilities for the system states are independent of starting conditions since all states are accessible from each other. Under this condition, the following "policy iteration" algorithm can be used to converge on an optimal assignment policy for each man-

Table 2.4: Possible Manpower Assignment Decisions

i	k	Manpower Assignment
1	1	*no maintenance reqd.*
2	1	$s_{21} = 1$
3	1	$s_{32} = 1$
4	1	$s_{42} = 1$
5	1	$s_{53} = 2,\ s_{54} = 1$
	2	$s_{53} = 2,\ s_{54} = 0$
	3	$s_{53} = 0,\ s_{54} = 1$
6	1	$s_{62} = 2$
	2	$s_{62} = 1$
7	1	$s_{73} = 1,\ s_{76} = 1$
	2	$s_{73} = 1,\ s_{76} = 0$
	3	$s_{73} = 0,\ s_{76} = 1$
8	1	$s_{84} = 1,\ s_{86} = 2$
	2	$s_{84} = 1,\ s_{86} = 0$
	3	$s_{84} = 0,\ s_{86} = 2$
9	1	$s_{95} = 1,\ s_{97} = 2,\ s_{98} = 1$
	2	$s_{95} = 1,\ s_{97} = 2,\ s_{98} = 0$
	3	$s_{95} = 1,\ s_{97} = 0,\ s_{98} = 1$
	4	$s_{95} = 1,\ s_{97} = 0,\ s_{98} = 0$
	5	$s_{95} = 0,\ s_{97} = 2,\ s_{98} = 1$
	6	$s_{95} = 0,\ s_{97} = 2,\ s_{98} = 0$
	7	$s_{95} = 0,\ s_{97} = 0,\ s_{98} = 1$
10	1	$s_{107} = 2$
	2	$s_{107} = 1$
11	1	$s_{117} = 2,\ s_{118} = 1$
	2	$s_{117} = 1,\ s_{118} = 0$
	3	$s_{117} = 0,\ s_{118} = 1$
12	1	$s_{129} = 1,\ s_{1210} = 2,\ s_{1211} = 1$
	2	$s_{129} = 1,\ s_{1210} = 2,\ s_{1211} = 0$
	3	$s_{129} = 1,\ s_{1210} = 0,\ s_{1211} = 1$
	4	$s_{129} = 1,\ s_{1210} = 0,\ s_{1211} = 0$
	5	$s_{129} = 0,\ s_{1210} = 2,\ s_{1211} = 1$
	6	$s_{129} = 0,\ s_{1210} = 2,\ s_{1211} = 0$
	7	$s_{129} = 0,\ s_{1210} = 0,\ s_{1211} = 1$
13	1	$s_{138} = 4$
	2	$s_{138} = 2$
14	1	$s_{149} = 2,\ s_{1411} = 2,\ s_{1413} = 1$
	2	$s_{149} = 2,\ s_{1411} = 2,\ s_{1413} = 0$
	3	$s_{149} = 2,\ s_{1411} = 0,\ s_{1413} = 1$
	4	$s_{149} = 2,\ s_{1411} = 0,\ s_{1413} = 0$
	5	$s_{149} = 0,\ s_{1411} = 2,\ s_{1413} = 1$
	6	$s_{149} = 0,\ s_{1411} = 2,\ s_{1413} = 0$
	7	$s_{149} = 0,\ s_{1411} = 0,\ s_{1413} = 1$
15	1	$s_{1512} = 4,\ s_{1514} = 2$
	2	$s_{1512} = 4,\ s_{1514} = 1$
	3	$s_{1512} = 4,\ s_{1514} = 0$
	4	$s_{1512} = 2,\ s_{1514} = 2$
	5	$s_{1512} = 2,\ s_{1514} = 1$
	6	$s_{1512} = 2,\ s_{1514} = 0$
	7	$s_{1512} = 0,\ s_{1514} = 2$
	8	$s_{1512} = 0,\ s_{1514} = 1$

Table 2.5: Nondominated Feasible Decisions (D_{gi})

	g				
i	1	2	3	4	5
1	{1}	{1}	{1}	{1}	{1}
2	{1}	{1}	{1}	{1}	{1}
3	{1}	{1}	{1}	{1}	{1}
4	{1}	{1}	{1}	{1}	{1}
5	{1}	{1}	{2,3}	{1}	{1}
6	{1}	{2}	{1}	{2}	{1}
7	{1}	{1}	{1}	{1}	{1}
8	{1}	{1}	{1}	{1}	{1}
9	{1}	{1}	{2,3}	{1}	{2,3,5}
10	{2}	{1}	{1}	{1}	{1}
11	{1}	{1}	{2,3}	{1}	{1}
12	{2,5}	{1}	{3,6}	{2,3,5}	{2,3,5}
13	{2}	{2}	{2}	{2}	{2}
14	{3,5}	{3,5}	{4,6,7}	{3,5}	{3,5}
15	{5}	{4}	{6,7}	{5,7}	{5,7}

power structure g (see Appendix A):

1. *Initialization.* For each state i and each decision $k \in D_{g_i}$, compute transition rate values r_{ij}^k as follows:

$$r_{ij}^k = \begin{cases} n_0^{(i)} q_{z^+(ij)} \mu_0 & \gamma(ij) = 0 \\ [s_{ij}^k / a_{\gamma(ij)}] \mu_{\gamma(ij)} & \gamma(ij) \in \{1, 2, \ldots, M\} \end{cases}$$
$$r_{ii}^k = -\sum_{j \neq i} r_{ij}^k$$

   Select an initial assignment policy (a decision k' for each state i).

2. *Value Determination.* Let v_i be the relative value of occupying a particular state i under the current policy. Use $r_{ij}^{k'}$ for the current policy to solve the set of equations

$$E(n_0) = n_0^{(i)} + \sum_{j=1}^{I} r_{ij}^{k'} v_j \quad i = 1, 2, \ldots, I \tag{2.20}$$

   for all relative values v_i and unknown $E(n_0)$ by arbitrarily setting v_I to zero.

3. *Policy Improvement.* For each state i, find the decision $k'' \in D_{g_i}$ that maximizes the expression $\sum_{j=1}^{I} r_{ij}^k v_j$ using the relative values of the current policy. If k'' is unchanged from k' for all states, stop with the optimal policy. Otherwise, decision k'' becomes the new current decision, so each $r_{ij}^{k'}$ is set equal to $r_{ij}^{k''}$. Return to Step 2.

The number of iterations required by the algorithm can be minimized by choosing a "good" initial policy. A suitable initial policy is fortunately represented by a set of "greedy" assignment decisions; that is, decisions which always favor the movement of machines which are closest to an operating condition. Such a policy results from

a left-to-right assignment of manpower to tasks as arrayed in Table 2.1. Transitions could be similarly arrayed for a system of any size. Table 2.3 presents the number of iterations required to converge on optimal assignment policies when this initialization rule is applied for the "Mike's Flying Club" example. Like the enumeration of a particular policy, each iteration of the policy iteration algorithm involves the solution of I simultaneous equations. Thus, the algorithm requires about $8/78 = 10.3\%$ of the computational effort required by exhaustive enumeration.

Application of the policy iteration algorithm to the example problem yields the results shown in the last two columns of Table 2.3. The table displays the expected number of operating aircraft and sortie generation rate achievable with each manpower structure. It is apparent that Mike could improve the performance of his enterprise by replacing his five specialized mechanics with three more costly mechanics qualified in all types of maintenance ($g = 5$). This alternative would produce a sortie generation rate of 5.045, which is higher than the maximum rate of 4.848 achievable with the current manpower structure ($g = 1$). The optimal manpower structure represents a fully "generalized" specialization strategy.

2.3 Linear Programming Model

While the policy iteration algorithm eliminates the need to enumerate solutions for every assignment policy, it still requires a complete solution for each manpower structure. Assignment decisions and manpower structures can be considered simultaneously through a linear programming formulation of the decision model. Linear programming (LP) provides an established method for solving Markov decision problems with finite state spaces [27], and the general approach can be efficiently applied to the manpower specialization problem.

2.3.1 Concurrent Approach

All candidate manpower structures and assignment policies can be evaluated concurrently in a single linear program. First, decision variables p_{gik} are defined as the joint probabilities of employing structure $g \in \{1, 2, \ldots, G\}$, finding the system in state $i \in \{1, 2, \ldots, I\}$, and selecting assignment decision $k \in D_{gi}$. An LP model can then be written as follows:

Maximize

$$E(n_0) = \sum_{g=1}^{G} \sum_{i=1}^{I} \sum_{k \in D_{gi}} n_0^{(i)} p_{gik} \tag{2.21}$$

subject to

$$\sum_{k \in D_{gi}} p_{gik} \left\{ n_0^{(i)} \mu_0 + \sum_{j=1}^{I} [\frac{s_{ij}^k}{a_{\gamma(ij)}}] \mu_{\gamma(ij)} \right\} = \sum_{j=1}^{I} \sum_{k \in D_{gj}} p_{gjk} \left\{ n_0^{(j)} q_{z^+(ji)} \mu_0 + [\frac{s_{ji}^k}{a_{\gamma(ji)}}] \mu_{\gamma(ji)} \right\}$$

$$g = 1, 2, \ldots, G \quad i = 1, 2, \ldots, I - 1 \tag{2.22}$$

$$\sum_{g=1}^{G} \sum_{i=1}^{I} \sum_{k \in D_{gi}} p_{gik} = 1 \tag{2.23}$$

$$p_{gik} \geq 0 \qquad g = 1, 2, \ldots, G \quad i = 1, 2, \ldots, I \quad k \in D_{gi} \tag{2.24}$$

The constraints represented by Equations 2.22 ensure that global balance for the queuing system is satisfied. Equation 2.23 is a normalization constraint for conservation of probability. Excluding the non-negativity restrictions imposed by Equations 2.24, the formulation will generate $G(I-1)+1$ constraints. The number of decision variables p_{gik} will depend on the total number of nondominated feasible

assignment decisions for all system states and manpower structures. For example, formulation of the "Mike's Flying Club" problem requires 71 constraints and 95 decision variables. When this model is implemented on a microcomputer using a standard software package [6], a solution is obtained in only a few seconds of processing time. The following decision variables have non-zero optimal values:

$p_{511} = .1803$	$p_{562} = .0445$	$p_{5113} = .0399$
$p_{521} = .1803$	$p_{573} = .0678$	$p_{5125} = .0342$
$p_{531} = .1564$	$p_{583} = .0491$	$p_{5131} = .0263$
$p_{541} = .0951$	$p_{595} = .0342$	$p_{5145} = .0168$
$p_{553} = .0484$	$p_{5102} = .0219$	$p_{5153} = .0047$

The example results provide a useful vehicle for examining some important characteristics of the LP solution. First, it should be noted that there is exactly one non-zero decision variable for each system state. The non-zero variables therefore represent the steady-state probabilities π_i. They also identify the optimal assignment decisions. While the model permits randomization between decisions, the optimal solution will force all probability for a given state to the decision which produces the largest rate of increase in the objective function. Similarly, only one manpower structure is represented in the final solution ($g = 5$), even though the model permits randomization between structures. The global balance equations ensure that all state probabilities for a given structure and policy maintain constant relative proportionate values. Therefore, the optimal solution will force all probability into the best structure.

The concurrent LP provides a compact representation of any manpower specialization problem. The model and supporting data can be easily formatted in a generalized algebraic language for computer implementation, so the formulation

can be useful for small problems. When solved using a standard simplex or revised simplex algorithm, the concurrent formulation will normally require a smaller number of LP iterations than would be required if the problem were decomposed to find separate optimal solutions for each manpower structure. However, each LP iteration for the concurrent formulation will require a much larger number of arithmetic operations. Thus, the concurrent formulation has limited practical value since a decomposed version of the problem will generally require less computational effort.

Issues concerning computational storage and analytical insight also impact the practical utility of the concurrent approach. Every candidate manpower structure g generates $I-1$ constraints and at least $I-1$ decision variables, so a problem involving a large number of structures and system states could easily exceed computational storage limits. Furthermore, while solution of the concurrent LP yields an optimal structure and associated system performance, it does not provide direct insight on how the optimal specialization strategy compares with others under consideration.

2.3.2 Sequential Approach

The disadvantages of the concurrent formulation can be overcome through a new algorithm which sequentially evaluates all candidate manpower structures but benefits from commonality of arithmetic operations. First, a single LP is formulated which permits any assignment decision that is feasible for at least one structure under consideration. Solution of this LP will produce an upper bound on the optimal system performance. Then, for each individual manpower structure, a constraint is added which forces all infeasible decision variables to zero. Integration of the new constraint into the linear programming tableau renders the original solution infeasible. However, feasibility can be restored by performing dual simplex

iterations [26]. These iterations are terminated when either a feasible (optimal) solution is reached or the objective value falls below a known lower bound for the specialization strategy employed by the current structure. This lower bound is established by the best objective value from previously evaluated structures which employ the same specialization strategy.

The algorithm can be accelerated by evaluating all manpower structures in order of their estimated relative performance (best to worst). A good estimate for relative performance of a structure g can be obtained from the parameter

$$\theta_g = \sum_{i=1}^{I} \left\{ \sum_{j=1}^{I} [\frac{s_{ij}^{k'}}{a_{\gamma(ij)}}] \mu_{\gamma(ij)} \right\}^{-1} \tag{2.25}$$

where $k' = \min(k \in D_{gi})$. A structure with a low value of θ_g will tend to produce favorable system performance since maintenance-intensive states will have relatively short transition times (inverses of transition rates). The ranking derived from θ_g is only an estimate since "greedy" (not necessarily optimal) assignment decisions are assumed and transition times are not weighted by steady-state probabilities.

The sequential linear programming algorithm can be concisely stated as follows:

1. For each system state i, define a set of decisions $D_i = \cup_{g=1}^{G} D_{gi}$. Let p_{ik} be the joint probability of finding the system in state i and selecting decision $k \in D_i$. Formulate and solve the bounding LP,

   Maximize

$$E(n_0) = \sum_{i=1}^{I} \sum_{k \in D_i} n_0^{(i)} p_{ik} \tag{2.26}$$

subject to

$$\sum_{k\in D_i} p_{ik}\left\{n_0^{(i)}\mu_0 + \sum_{j=1}^{I}[\frac{s_{ij}^k}{a_{\gamma(ij)}}]\mu_{\gamma(ij)}\right\} = \sum_{j=1}^{I}\sum_{k\in D_j} p_{jk}\left\{n_0^{(j)}q_{z^+(ji)}\mu_0 + [\frac{s_{ji}^k}{a_{\gamma(ji)}}]\mu_{\gamma(ji)}\right\}$$

$$i = 1, 2, \ldots, I-1 \tag{2.27}$$

$$\sum_{i=1}^{I}\sum_{k\in D_i} p_{ik} = 1 \tag{2.28}$$

$$p_{ik} \geq 0 \qquad i = 1, 2, \ldots, I \quad k \in D_i \tag{2.29}$$

Store the linear programming tableau.

2. For each specialization strategy $h \in \{1, 2, \ldots, H\}$, define incumbent optimal performance values $E(n_0)_h$ with initial values of zero. Reorder all manpower structures $g \in \{1, 2, \ldots, G\}$ such that $\theta_1 \leq \theta_2 \leq \ldots \leq \theta_G$ where each θ_g is defined by Equation 2.25. For $g = 1, \ldots, G$,

   (a) Define a set of infeasible decisions $\bar{D}_{gi} = \{k : k \in D_i, k \notin D_{gi}\}$. Modify the LP tableau from Step 1 by adding the constraint

$$\sum_{i=1}^{I}\sum_{k\in \bar{D}_{gi}} p_{ik} = 0 \tag{2.30}$$

   (b) If the addition of Equation 2.30 results in an infeasible current solution, perform dual simplex iterations [2, p. 182] until $E(n_0) \leq E(n_0)_{h_g}$ or $E(n_0)$ is optimal. If $E(n_0)$ is optimal, then let $E(n_0)_{h_g} = E(n_0)$.

3. Identify an optimal manpower manpower specialization strategy h' such that $E(n_0)_{h'} \geq E(n_0)_h \; \forall \; h \in \{1, 2, \ldots, H\}$.

The sequential algorithm minimizes computational effort by taking advantage of state relationships which are common to different manpower structures and assignment decisions. For the "Mike's Flying Club" example, Step 1 of the algorithm requires 15 LP iterations to establish the initial bounding solution, and Step 2 requires $3 + 1 + 3 + 3 + 1 = 11$ iterations to evaluate the individual manpower structures. The 26 total iterations generate performance results for all three specialization strategies under consideration.

Two variations of the sequential algorithm may be useful for particular applications. In some cases, it may be desirable to gain additional computational speed in exchange for less analytical insight. This is accomplished by modifying Step 2 so that all incumbent optimal objective values $E(n_0)_h$ are replaced by a single global value $E(n_0)_{\text{opt}}$. For each manpower structure g, dual simplex iterations are performed until $E(n_0) \leq E(n_0)_{\text{opt}}$ or $E(n_0)$ is optimal. If $E(n_0)$ is optimal, it becomes the new $E(n_0)_{\text{opt}}$. The modified algorithm identifies only a global optimal manpower structure rather than an optimal structure for each specialization strategy. However, execution of this "speed" version of the algorithm can require a significantly smaller number of LP iterations.

Another variation of the algorithm can provide full analytical insight by completely evaluating all manpower structures. For every structure, dual simplex iterations are continued until an optimal objective value is achieved. No benefit is derived by evaluating structures in a particular order, so the reordering procedure can be deleted. However, more LP iterations are required by this "insight" version of the algorithm.

All three versions of the sequential linear programming algorithm (normal, speed, and insight) are implemented in the computer program described in Appendix B. Table 2.6 displays computational performance results (number of LP iterations) for each version when applied to three example problems of different sizes. Each iteration count includes all pivots to establish an initial "greedy" solution to the bounding LP, all primal pivots to optimize the bounding solution, and all dual pivots required by Step 2 of the algorithm. The iteration counts are compared against a baseline number of iterations which are required if separate linear programs are solved for each manpower structure. It is noteworthy that the normal version of the algorithm requires 28.1–33.8% of the baseline computational effort, and the speed version requires 13.4–27.9% of the baseline effort. The LP iteration counts correlate approximately with comparative run times for each problem size.

Table 2.6: Computational Performance of Sequential Algorithms

Problem	I	G	Number of LP Iterations			
			Baseline	Insight	Normal	Speed
Mike's Flying Club ($N = 2, C = 100$)	15	5	79	31	26	22
Mike's Flying Club ($N = 4, C = 200$)	70	8	613	291	172	122
YF-XX (see Chapter 3)	455	11	5618	2825	1897	774

2.4 Network Reduction Procedure

Even when manpower structures are considered sequentially, problem dimensionality can still be an important issue. For example, consider a system with three machines and six task types, where all tasks can be accomplished simultaneously. The corresponding queuing network will have $Z = \sum_{m=1}^{6} \binom{6}{m} = 255$

maintenance conditions, so the system will have $I = \binom{3+255}{255} = 2.829 \times 10^6$ states. Fortunately, the routing probabilities and average service times associated with many of the maintenance conditions could be very small, so a very good approximate solution could be obtained by "collapsing" these conditions into others. The notion of flow equivalence can be applied to form a new reduced network that will approximate the behavior of the original network.

Like all previous concepts, the network reduction approach can be demonstrated using "Mike's Flying Club." Suppose computational facilities are so limited that any system with more than ten states is intractable. It is therefore necessary to develop a new network that will approximate the original example network while reducing the total number of system states. This can be done by eliminatng station 4 and adjusting the characteristics of the stations which can be entered directly from station 4 (stations 2 and 3). The adjusted characteristics include the routing probabilities and service rates. The resulting reduced network will have three maintenance stations and $\binom{5}{3} = 10$ states.

Routing probability can be conserved by adjusting q_2 and q_3 to new values q_2' and q_3' as follows:

$$\begin{aligned} q_2' &= q_2 + q_4 \left(\frac{\mu_3}{\mu_2 + \mu_3} \right) \\ &= .1404 + (.1454) \left(\frac{.5}{.25 + .5} \right) = .2373 \end{aligned}$$

$$\begin{aligned} q_3' &= q_3 + q_4 \left(\frac{\mu_2}{\mu_2 + \mu_3} \right) \\ &= .1880 + (.1454) \left(\frac{.25}{.25 + .5} \right) = .2364 \end{aligned}$$

This operation distributes q_4 to succeeding stations in proportion to the relative

probabilities of machine entry from station 4. It is now necessary to decrease the service rates at stations 2 and 3 to reflect the longer service times for machines requiring both types of maintenance:

$$\begin{aligned}
\mu_2' &= \left\{ \frac{q_2\left(\frac{1}{\mu_2}\right) + q_4\left(\frac{\mu_3}{\mu_2+\mu_3}\right)\left(\frac{1}{\mu_2}+\frac{1}{\mu_2+\mu_3}\right)}{q_2 + q_4\left(\frac{\mu_3}{\mu_2+\mu_3}\right)} \right\}^{-1} \\
&= q_2'\left\{ q_2\left(\frac{1}{\mu_2}\right) + q_4\left(\frac{\mu_3}{\mu_2+\mu_3}\right)\left(\frac{1}{\mu_2}+\frac{1}{\mu_2+\mu_3}\right) \right\}^{-1} \\
&= \frac{.2373}{1.0785} = .2200
\end{aligned}$$

$$\begin{aligned}
\mu_3' &= \left\{ \frac{q_3\left(\frac{1}{\mu_3}\right) + q_4\left(\frac{\mu_2}{\mu_2+\mu_3}\right)\left(\frac{1}{\mu_3}+\frac{1}{\mu_2+\mu_3}\right)}{q_3 + q_4\left(\frac{\mu_2}{\mu_2+\mu_3}\right)} \right\}^{-1} \\
&= q_3'\left\{ q_3\left(\frac{1}{\mu_3}\right) + q_4\left(\frac{\mu_2}{\mu_2+\mu_3}\right)\left(\frac{1}{\mu_3}+\frac{1}{\mu_2+\mu_3}\right) \right\}^{-1} \\
&= \frac{.2364}{.5376} = .4400
\end{aligned}$$

The reduction in service rates compensates for the deletion of station 4.

This example reduction can be expanded into a general procedure. First, let $I' < I$ be a maximum number of system states imposed by computational limitations. The maximum number of reduced network stations can therefore be represented as $Z' < Z$, where $\binom{N+Z'}{Z'} \leq I' < \binom{N+Z'+1}{Z'+1}$. If all maintenance stations are ordered by increasing numbers of tasks in their pending task sets, a network reduction procedure can be constructed as follows:

1. Define station-dependent task rates $\mu_m^{(z)}$ for each station and applicable task. Initially, let $\mu_m^{(z)} = \mu_m$ for each $z \in \{1, 2, \ldots, Z\}$ and each eligible task $m \in E_z$. Define z^- as the current station to be removed from the network. Initially, let $z^- = Z$.

2. Let S_{z^-} be the set of stations which can be directly entered from station z^-. For each station $z^+ \in S_{z^-}$,

   (a) Identify the task $\epsilon \in E_{z^-}$ which causes machine movement to station z^+ on completion.

   (b) Compute a new routing probability as

$$q'_{z^+} = q_{z^+} + q_{z^-} \left(\frac{\mu_\epsilon^{(z^-)}}{\sum_{m \in E_{z^-}} \mu_m^{(z^-)}} \right) \tag{2.31}$$

   (c) For each task $m \in E_{z^+}$, compute a new service rate as

$$\begin{aligned} \mu'_m{}^{(z^+)} = & \left(\frac{\mu_m^{(z^+)}}{\sum_{\eta \in E_{z^+}} \mu_\eta^{(z^+)}} \right) q'_{z^+} \left\{ q_{z^+} \left(\frac{1}{\sum_{\eta \in E_{z^+}} \mu_\eta^{(z^+)}} \right) \right. \\ & \left. + q_{z^-} \left(\frac{\mu_\epsilon^{(z^-)}}{\sum_{\eta \in E_{z^-}} \mu_\eta^{(z^-)}} \right) \left(\frac{1}{\sum_{\eta \in E_{z^+}} \mu_\eta^{(z^+)}} + \frac{1}{\sum_{\eta \in E_{z^-}} \mu_\eta^{(z^-)}} \right) \right\}^{-1} \end{aligned} \tag{2.32}$$

3. For each z^+ in S_{z^-}, let $q_{z^+} = q'_{z^+}$ and $\mu_m^{(z^+)} = \mu'_m{}^{(z^+)}$ $\forall$ $m \in E_{z^+}$. Let $z^- = z^- - 1$. If $z^- = Z'$, then stop with the final reduced network. Otherwise, return to Step 2.

This procedure will transfer maintenance time at stations with many pending tasks to stations with fewer pending tasks. Consequently, the reduced network will not capture the queuing delays caused by resource conflicts between the eligible tasks at each eliminated station. The approximation will therefore overestimate system performance. However, an approximate flow equivalent network can provide very accurate results unless multiple task eligibilities are very common and resource availability is very low. When solving the example problem using the reduced network, an optimal sortie generation rate of 5.130 sorties per aircraft per

day is obtained. This represents an error of only 1.75%, even though a sizable portion (.1454) of the network routing probability is redistributed. More importantly, the optimal manpower structure is unchanged from the full network solution, $\vec{x} = (0, 0, 0, 0, 3)$.

Chapter 3

Application

In the previous chapter, a simple example was used to develop a queuing network approach to maintenance manpower specialization. In this chapter, the method is applied to a larger problem which is more representative of potential "real world" applications. The problem objective is to derive an optimal manpower structure for dispersed operation of a notional military aircraft.

3.1 The YF-XX Aircraft Maintenance Problem

The YF-XX is a notional tactical fighter in the prototype phase of weapon system development. Since the earliest stages of the system design, significant effort has been focused on reliability and maintainability characteristics. Consequently, substantial analysis has been conducted to estimate subsystem failure rates, repair times, and task personnel requirements. Since extensive test data has not yet been accumulated, precise forms for probability distributions of failure and repair times are not known. However, efforts to eliminate known failure mechanisms suggest that subsystem malfunctions can be regarded as "random" and can thus be accurately represented by exponentially distributed failure times. The exponential distribution is also postulated as a suitable model for maintenance task time at the major subsystem level. The aggregate maintenance data for the YF-XX are shown in Table 3.1.

The operational concept for the YF-XX involves a wide variety of mission profiles with an average duration of 1.6 hours ($\mu_0 = .625$). Turn-around mainte-

Table 3.1: Task Data for YF-XX Tactical Fighter

m	Task Description	λ_m	μ_m	a_m
1	Munitions Upload	–	2.400	3
2	Aircraft Turn-Around	–	2.000	1
3	Avionics Repair	.0713	.4191	1
4	General Aircraft Repair	.0269	.2390	2
5	Engine Repair	.0250	.2301	2
6	Electrical Subsystem Repair	.0232	.2107	1
7	Pneudraulic Subsystem Repair	.0144	.4223	2
8	Fuel Subsystem Repair	.0103	.1812	2
9	Armament Subsystem Repair	.0091	.3200	2

nance and munitions upload are accomplished between all sorties. All unscheduled maintenance generated by subsystem malfunctions must be performed prior to aircraft turn-around, and turn-around must be completed before munitions upload. Additionally, all electrical subsystem repair must precede any work on the aircraft avionics (communications, navigation, electronic counter-measures, etc.). Figure 3.1 displays a reduced network which captures these requirements. The figure also displays the adjusted routing probabilities associated with each network station. While the unreduced network included 129 maintenance stations, less than 2.5% of the total routing probability had to be redistributed among stations 3 through 12 using the procedure described in Section 2.4.

3.2 Maintenance Cost Model

The basic maintenance concept for the YF-XX assigns the nine task types to seven manpower specialties. However, wartime operation plans dictate that a significant number of aircraft will operate in groups of three at dispersed locations. It is anticipated that some maintenance specialty merger will be required for dis-

z	1	2	3	4	5	6	7	8	9	10	11	12
P_z	{1}	{1,2}	{1.2.3}	{1.2.4}	{1.2.5}	{1.2.6}	{1.2.7}	{1.2.8}	{1.2.9}	{1.2.3.4}	{1.2.3.5}	{1.2.3.6}
E_z	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}	{9}	{3.4}	{3.5}	{6}
q'_z	0.0	.7762	.0787	.0304	.0284	.0262	.0173	.0137	.0113	.0063	.0058	.0059

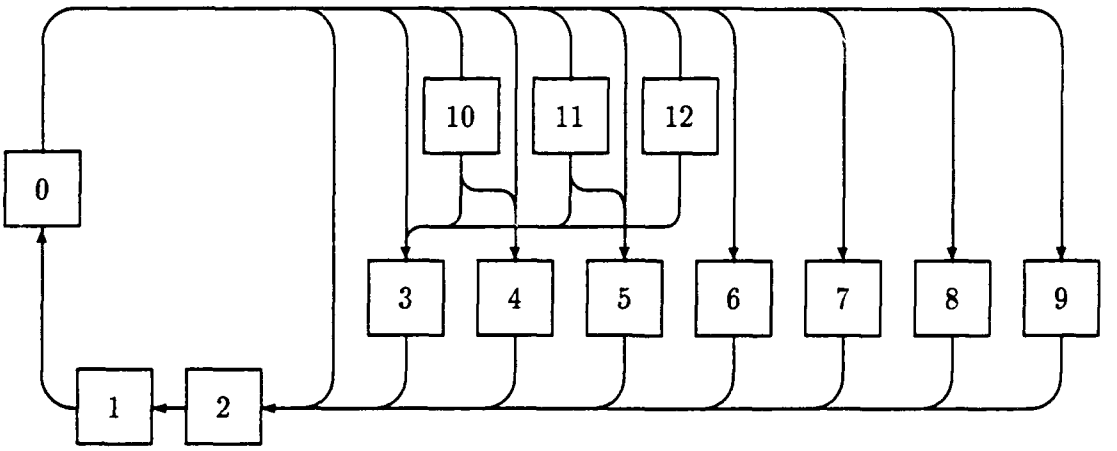


Figure 3.1: Reduced Network for YF-XX Aircraft Maintenance

persed forces in order to satisfy manpower cost constraints. Table 3.2 lists thirteen types of specialties which appear in various consolidation schemes under consideration. The table displays the task qualifications for each specialty and provides additional information that can be used to develop cost estimates. The estimated length (in days) of the initial training period for each type of specialty is indicated as L_y. All training generates a direct cost of \$200 per person per day. Also shown is the expected monetary bonus V_y (in dollars) which would be paid to each type of specialist if he were to reenlist after a four-year term of service. This value is largely determined by the demand for personnel with similar skills in the civilian economy.

An expected profile of the entire maintenance workforce is presented in Figure 3.2. This profile is adapted from a demonstrative model used in the Air Force SUMMA project [28, p. 10]. The model asserts that about 60% of the workforce will

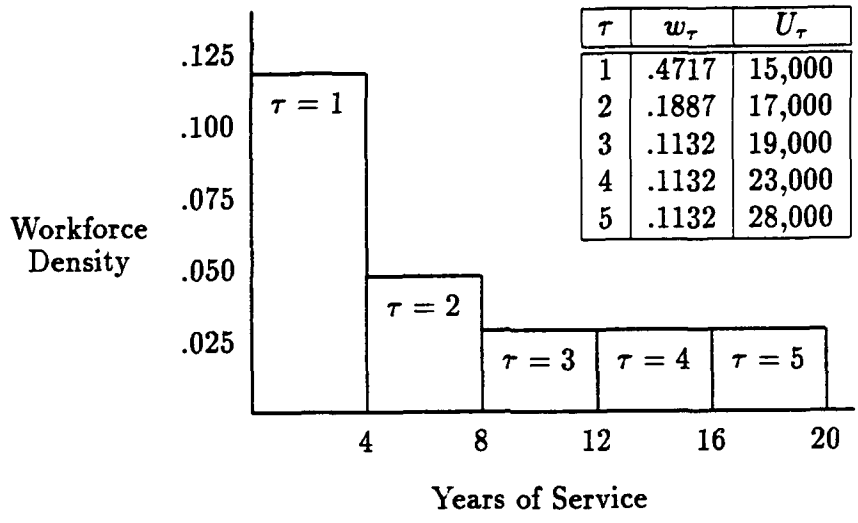
Table 3.2: Maintenance Specialties for YF-XX

y	Q_y	L_y	V_y	c_y
1	{1,9}	90	–	20,934
2	{2,4}	100	5,000	21,637
3	{3}	120	10,000	22,679
4	{5}	120	5,000	22,302
5	{6}	95	10,000	21,851
6	{7}	90	–	20,934
7	{8}	90	–	20,934
8	{3,6}	180	10,000	24,799
9	{5,8}	150	5,000	23,338
10	{5,7,8}	200	5,000	25,174
11	{1,2,4,9}	150	5,000	23,338
12	{1,2,4,5,7,8,9}	270	12,000	28,531
13	{1,2,3,4,5,6,7,8,9}	360	15,000	32,924

depart after a single four-year term of service, and 40% of the remaining workforce will depart after a second term of service. Bonuses are to be paid for second and third term reenlistments in order to restrict workforce attrition to these levels for all specialties. It is further postulated that all personnel who have completed three terms of service will remain until retirement after a 20-year career. These parameters establish portions of the workforce w_τ for each term $\tau \in \{1, 2, \ldots, 5\}$. The average annual cost of pay and benefits U_τ for personnel in each term is also shown. Pay and benefits do not vary with specialty.

Training costs, retention costs, and the information presented in Figure 3.2 can be aggregated to produce total annual costs for each specialty as follows:

$$\begin{aligned} c_y &= w_1 \left(\frac{(4)(365)}{(4)(365) - L_y} \right) \left(\frac{200}{4} L_y + U_1 \right) \\ &\quad + w_2 \left(U_2 + \frac{V_y}{4} \right) + w_3 \left(U_3 + \frac{V_y}{4} \right) + w_4 U_4 + w_5 U_5 \end{aligned} \tag{3.1}$$



τ	w_τ	U_τ
1	.4717	15,000
2	.1887	17,000
3	.1132	19,000
4	.1132	23,000
5	.1132	28,000

Figure 3.2: Profile of YF-XX Manpower Force

The aggregate cost parameters listed in the last column of Table 3.2 are computed using this formula. Note the the first term of the equation includes an expression which accounts for the opportunity cost of training time. Clearly, training and retention factors cause a significant increase in the unit cost of maintenance personnel as specialization decreases. Table 3.2 indicates that the annual cost of a maintenance technician with a full range of skills is roughly 50% more than that of a typical technician with a highly specialized skill.

3.3 Manpower Optimization

The annual maintenance manpower budget for the YF-XX is constrained to a level of \$120,000 per aircraft. Since wartime plans assume that each maintenance technician will be on duty for a 12-hour shift daily, the total expenditure limit per shift at each dispersal base can be calculated as $C = (3)(12/24)(120,000) = 180,000$. As indicated in Table 3.3, this constraint translates to a total of 11 candidate manpower structures. Note that the expenditure limit will not support a

fully specialized structure. Several specialty types do not appear in any of the acceptable alternatives.

Table 3.3: Manpower Structures for YF-XX

g	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	$\sum c_y x_y$	$E(n_0)$	R
1	0	0	1	0	1	0	0	0	0	0	2	3	0	164,892	1.271	6.353*
2	0	0	1	0	1	0	0	0	0	0	0	0	4	158,654	1.308	6.539*
3	0	0	1	0	3	0	0	0	0	0	0	0	3	173,825	1.216	6.080
4	0	0	2	0	2	0	0	0	0	0	0	0	3	174,653	1.223	6.115
5	0	0	3	0	1	0	0	0	0	0	0	0	3	175,481	1.220	6.100
6	0	0	0	0	0	0	0	1	0	2	4	0	0	168,499	1.319	6.593*
7	0	0	0	0	0	0	0	2	0	2	3	0	0	169,960	1.283	6.415
8	0	0	0	0	0	0	0	1	0	0	0	5	0	167,454	1.345	6.724*
9	0	0	0	0	0	0	0	2	0	0	0	4	0	163,722	1.322	6.610
10	0	0	0	0	0	0	0	3	0	0	0	3	0	159,990	1.223	6.115
11	0	0	0	0	0	0	0	0	0	0	0	0	5	164,620	1.369	6.844**

* local optimum for specialization strategy
** global optimum

Table 3.3 also displays the performance results obtained when queuing network analysis is applied. The reduced network shown in Figure 3.1 generates $\binom{3+12}{12} = 455$ system states, so the problem is quite tractable when solved using the sequential linear programming algorithm. Results are shown for each manpower structure, indicating a steady improvement in sortie generation capability as specialization decreases. Dispersal bases will operate most effectively if each maintenance shift is manned by five personnel that are qualified on all aircraft systems. Structure 8 also produces good results and might be favored in the context of broader manpower issues. This structure employs one avionics/electrical system specialist, and five technicians qualified in all other types of maintenance.

Chapter 4

Extensions

The general model presented in this thesis can be adapted to capture a variety of maintenance concepts which might arise in specific applications. In this chapter, the versatility of the model is demonstrated by extending it to address some issues which might apply to the aircraft maintenance problem represented by the "Mike's Flying Club" example.

4.1 Cross-Training

It has been shown that a queuing network model can be used to optimize the basic structure of a maintenance manpower force. This same approach can be extended to determine optimal "cross-training" for maintenance specialists so that they possess secondary skills.

Suppose the "Mike's Flying Club" enterprise currently employs a fully specialized maintenance force consisting of two turn-around mechanics, one airframe mechanic, and two engine mechanics. Maintenance specialists with broader skills are not immediately available, but Mike has the opportunity to provide some employees (current or future) with additional training. Specifically, he can qualify one or more airframe mechanics to assist an engine specialist with engine repairs. The cost of this training would translate into an incremental expense of $10 per hour. In addition, cross-trained airframe mechanics could be further upgraded to perform turn-around functions at an incremental cost of only $3 per hour.

The cross-training options selected for this example result in a set of po-

tential maintenance specialties with unit costs and task qualifications which are identical to those used in earlier analysis (see Table 2.2). However, the concept of cross-training produces a very different set of acceptable manpower structures. The earlier requirement that a maintenance task can be allocated to only one type of specialist is now eliminated. Instead, an acceptable manpower structure must satisfy the following constraints:

1. Total hourly manpower expenditure must not exceed the fixed budget; i.e., $\sum_{y=1}^{5} c_y x_y \leq 100$.

2. Sufficient mechanics must be qualified to perform or assist in each type of task; i.e., $\sum_{y:m \in Q_y} x_y \geq a_m \; \forall \, m \in \{1,2,3\}$.

3. Sufficient primary-skilled mechanics must be available for each type of task requiring multiple personnel; i.e., $x_3 \geq 1$ (at least one engine mechanic must be available).

4. All skills must have a potential for utilization; i.e., $\sum_{y:m \in Q_y} x_y \leq a_m N \; \forall \, m \in \{1,2,3\}$.

These constraints, in general form, will apply to a problem of arbitrary size. The fact that cross-trained specialists may not be as effective in their secondary skills as they are in their primary skills can be reflected in lower task completion rates. For this example, assume that an engine mechanic and cross-trained airframe mechanic can accomplish engine repairs at a slightly degraded rate of .45 tasks per hour. Similarly, assume that a cross-trained airframe mechanic can accomplish turn-around tasks at an average rate of .9 per hour. Table 4.1 summarizes the rate capabilities for each type of specialist and lists all of the candidate manpower structures.

Table 4.1: Manpower Structures with Cross-Training

g	x_1	x_2	x_3	x_4	x_5	$E(n_0)$	R
1	2	1	2	0	0	.8080	4.848
2	1	2	2	0	0	.8159	4.895
3	2	1	1	1	0	.8322	4.993
4	2	0	2	1	0	.8080	4.848
5	1	1	1	0	1	.8302	4.981
6	1	0	2	0	1	.8062	4.837
7	1	0	1	2	0	.7970	4.782
8	1	0	1	1	1	.8302	4.981
9	0	0	1	0	2	.8065	4.839
μ_1	1.0				0.90		
μ_2		0.25		0.25	0.25		
μ_3			0.50	0.45	0.45		

The policy iteration or linear programming optimization methods can be applied to solve a cross-training problem with minor adaptation. The set of manpower assignment decisions for each state may expand, since some decisions may need to reflect a choice of which specialist type(s) are assigned to each transition task. Transition rates are determined not only by manpower assignment variables s_{ij}^k, but also by corresponding task completion rates $\mu_{\gamma(ij)}^k$.

The last two columns of Table 4.1 display the performance results for the "Mike's Flying Club" problem with cross-training. The optimal sortie generation rate of 4.993 is achieved with manpower structure 3, where $\vec{x} = (2,1,1,1,0)$. Mike should therefore dismiss an engine mechanic, hire another airframe mechanic, and provide cross-training for one airframe mechanic in engine repairs. This new manpower structure will improve his average sortie generation rate by about 3% over the original rate achieved with structure 1.

4.2 Dependent Machine Operations

Another modeling problem which may be encountered in applications is a need to operate two or more machines together (e.g., flights of multiple aircraft). Suppose half of the flights scheduled in the "Mike's Flying Club" enterprise are acrobatic formation flights which require two aircraft. This operational requirement can be modeled using the network shown in Figure 4.1. The network station 0^1 represents the operating condition for single aircraft flights, while station 0^2 represents the operating condition for formation flights. Station 0^2 is also used to hold a single aircraft that has completed maintenance and is waiting for a companion aircraft. In general, a network station 0^b is required for every group size $b \in B$, where B is the set of all required group sizes ($B = \{1, 2\}$ for the example). The expected number of operating machines can be computed as $\sum_{i=1}^{I} b[n_{0^b}^{(i)}/b]\pi_i$.

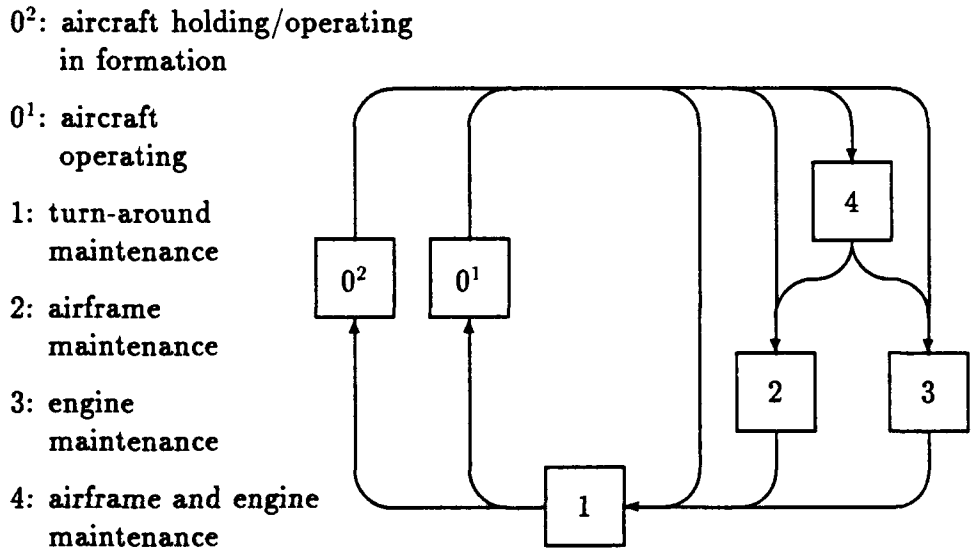


Figure 4.1: Dependent Aircraft Operation

Table 4.2 describes the 21 states for the example system with $N = 2$ aircraft and illustrates the feasible state transitions. Transition activities for the lower right portion of the table are identical to those for the example without dependent aircraft operations (see Table 2.1). However, transition rates involving maintenance task 1 now reflect the different stations which an aircraft can enter from station 1. Define $\rho^1 = .5$ as the probability that the next scheduled flight requires a single aircraft. Similarly, define $\rho^2 = .5$ as the probability that the next scheduled flight requires two aircraft. Transition rates can be expressed as $[s_{ij}^k / a_{\gamma(ij)}]\rho^1 \mu_{\gamma(ij)}$ for $(i,j) \in \{(8,7),(12,8),(13,9),(14,10),(15,11)\}$ and as $[s_{ij}^k / a_{\gamma(ij)}]\rho^2 \mu_{\gamma(ij)}$ for $(i,j) \in \{(8,2),(12,3),(13,4),(14,5),(15,6)\}$. For the case where an aircraft is holding for a formation companion (state 3), probabilistic branching does not apply since the next aircraft leaving station 1 must enter station 0^2.

Table 4.2: Transition Activities (γ) for Dependent Operations

$\vec{n}(i)$		j																				
$0^2 0^1 1234$	i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
200000	1												0^2	0^2	0^2	0^2	0^2	0^2	0^2	0^2	0^2	0^2
110000	2			0^1	0^1	0^1	0^1															
101000	3	1																				
100100	4			2																		
100010	5			3																		
100001	6				3	2																
020000	7								0^1	0^1	0^1	0^1										
011000	8		1					1					0^1	0^1	0^1	0^1						
010100	9								2					0^1			0^1	0^1	0^1			
010010	10								3						0^1			0^1		0^1	0^1	
010001	11									3	2					0^1			0^1		0^1	0^1
002000	12			1					1													
001100	13				1					1			2									
001010	14					1					1		3									
001001	15						1					1		3	2							
000200	16													2								
000110	17													3	2							
000101	18															2	3	2				
000020	19														3							
000011	20															3		3		2		
000002	21																		3		2	

While a single gaining station $z^+(ij)$ was formerly associated with each transition, this parameter must now be replaced by a list of gaining stations $Z^+(ij)$. The routing probabilities q_{z^+} must also be displaced by new values $q^b_{Z^+}$ for each station 0^b. The computation of a routing probability $q^1_{Z^+}$ for a single aircraft completing a sortie is unchanged from previous analysis, but more effort is required to obtain probabilities for aircraft completing a formation flight. Let T_{0^2} be the duration of the flight, let $T1_2$ be the time until an airframe malfunction occurs for the first aircraft in the formation, and let $T1_3$ be the time until an engine malfunction occurs for the first aircraft. Similarly, let $T2_2$ and $T2_3$ represent the malfunction times for the second aircraft. Routing probabilities $q^2_{Z^+}$ can then be computed as follows:

$$q^2_{\{1,1\}} = P\{T_{0^2} < \min(T1_2, T1_3, T2_2, T2_3)\} = \frac{\mu_0}{\mu_0 + 2\lambda_2 + 2\lambda_3} = .3571$$

$$\begin{aligned} q^2_{\{1.2\}} &= P\{T_{0^2} < \min(T1_2, T1_3, T2_3),\ T_{0^2} \geq T2_2\} \\ &\quad + P\{T_{0^2} < \min(T1_3, T2_2, T2_3),\ T_{0^2} \geq T1_2\} \\ &= 2\left(\frac{\mu_0}{\mu_0 + \lambda_2 + 2\lambda_3} - q^2_{\{1,1\}}\right) = .1190 \end{aligned}$$

$$\begin{aligned} q^2_{\{1,3\}} &= P\{T_{0^2} < \min(T1_2, T2_2, T2_3),\ T_{0^2} \geq T1_3\} \\ &\quad + P\{T_{0^2} < \min(T1_2, T2_2, T1_3),\ T_{0^2} \geq T2_3\} \\ &= 2\left(\frac{\mu_0}{\mu_0 + 2\lambda_2 + \lambda_3} - q^2_{\{1,1\}}\right) = .1553 \end{aligned}$$

$$\begin{aligned} q^2_{\{1.4\}} &= P\{T_{0^2} < \min(T1_2, T1_3),\ T_{0^2} \geq \max(T2_2, T2_3)\} \\ &\quad + P\{T_{0^2} < \min(T2_2, T2_3),\ T_{0^2} \geq \max(T1_2, T1_3)\} \end{aligned}$$

$$= 2\left(\frac{\mu_0}{\mu_0+\lambda_2+2\lambda_3} - q^2_{\{1,1\}} - \frac{q^2_{\{1,2\}}}{2} - \frac{q^2_{\{1,3\}}}{2}\right) = .0641$$

$$q^2_{\{2,2\}} = P\{T_{0^2} < \min(T1_3, T2_3),\ T_{0^2} \geq \max(T1_2, T2_2)\}$$
$$= \frac{\mu_0}{\mu_0+2\lambda_3} - q^2_{\{1,1\}} - q^2_{\{1,2\}} = .0238$$

$$q^2_{\{2,3\}} = P\{T_{0^2} < \min(T1_2, T2_3),\ T_{0^2} \geq \max(T1_3, T2_2)\}$$
$$+ P\{T_{0^2} < \min(T2_2, T1_3),\ T_{0^2} \geq \max(T1_2, T2_3)\}$$
$$= 2\left(\frac{\mu_0}{\mu_0+\lambda_2+\lambda_3} - q^2_{\{1,1\}} - \frac{q^2_{\{1,2\}}}{2} - \frac{q^2_{\{1,3\}}}{2}\right) = .0641$$

$$q^2_{\{2,4\}} = P\{T_{0^2} < T1_3,\ T_{0^2} \geq \max(T1_2, T2_2, T2_3)\}$$
$$+ P\{T_{0^2} < T2_3,\ T_{0^2} \geq \max(T1_2, T1_3, T2_2)\}$$
$$= 2\left(\frac{\mu_0}{\mu_0+\lambda_3} - q^2_{\{1,1\}} - q^2_{\{1,2\}} - \frac{q^2_{\{1,3\}}}{2} - \frac{q^2_{\{1,4\}}}{2} - q^2_{\{2,2\}} - \frac{q^2_{\{2,3\}}}{2}\right) = .0500$$

$$q^2_{\{3,3\}} = P\{T_{0^2} < \min(T1_2, T2_2),\ T_{0^2} \geq \max(T1_3, T2_3)\}$$
$$= \frac{\mu_0}{\mu_0+2\lambda_2} - q^2_{\{1,1\}} - q^2_{\{1,3\}} = .0431$$

$$q^2_{\{3,4\}} = P\{T_{0^2} < T1_2,\ T_{0^2} \geq \max(T1_3, T2_2, T2_3)\}$$
$$+ P\{T_{0^2} < T2_2,\ T_{0^2} \geq \max(T1_2, T1_3, T2_3)\}$$
$$= 2\left(\frac{\mu_0}{\mu_0+\lambda_2} - q^2_{\{1,1\}} - \frac{q^2_{\{1,2\}}}{2} - q^2_{\{1,3\}} - \frac{q^2_{\{1,4\}}}{2} - q^2_{\{3,3\}} - \frac{q^2_{\{2,3\}}}{2}\right) = .0704$$

$$q^2_{\{4,4\}} = P\{T_{0^2} \geq \max(T1_2, T1_3, T2_2, T2_3)\}$$
$$= 1 - q^2_{\{1,1\}} - q^2_{\{1,2\}} - q^2_{\{1,3\}} - q^2_{\{1,4\}}$$
$$- q^2_{\{2,2\}} - q^2_{\{2,3\}} - q^2_{\{2,4\}} - q^2_{\{3,3\}} - q^2_{\{3,4\}} = .0532$$

Unlike previous versions of the queuing network model, the dependent machine extension does not exhibit one-step behavior. However, since the policy iteration and linear programming approaches are based on global balance, both solution methods can still be applied. For notational convenience, rates r_{ij}^k are defined for every possible system transition:

$$r_{ij}^k = \begin{cases} [n_{0^{b'}}/b']q_{Z^+(ij)}^{b'}\mu_{0^{b'}} & \exists\, b' \in B : z^-(ij) = 0^{b'} \\ [s_{ij}^k/a_{\gamma(ij)}]\rho^{b'}\mu_{\gamma(ij)} & \exists\, b' \in B : Z^+(ij) = \{0^{b'}\}, \\ & n_{0^b}^{(i)} - b[n_{0^b}^{(i)}/b] = 0 \;\forall\, b \in B \\ [s_{ij}^k/a_{\gamma(ij)}]\mu_{\gamma(ij)} & \text{otherwise} \end{cases}$$

An extended version of the concurrent linear programming formulation can then be written as follows:

Maximize

$$\sum_{g=1}^{G}\sum_{i=1}^{I}\sum_{k\in D_{gi}}\sum_{b\in B} b[\frac{n_{0^b}^{(i)}}{b}]p_{gik} \tag{4.1}$$

subject to

$$\sum_{k\in D_{gi}} p_{gik}\sum_{j=1}^{I} r_{ij}^k = \sum_{j=1}^{I}\sum_{k\in D_{gj}} p_{gjk}\, r_{ji}^k \qquad g = 1,2,\ldots,G \quad i = 1,2,\ldots,I-1 \tag{4.2}$$

$$\sum_{g=1}^{G}\sum_{i=1}^{I}\sum_{k\in D_{gi}} p_{gik} = 1 \tag{4.3}$$

$$p_{gik} \geq 0 \qquad g = 1,2,\ldots,G \quad i = 1,2,\ldots,I \quad k \in D_{gi} \tag{4.4}$$

Table 4.3 displays performance results for all manpower structures in the extended "Mike's Flying Club" example. It is assumed that average sortie durations are identical for single aircraft flights and formation flights ($\mu_{0^1} = \mu_{0^2} = .5$), although this is not required in the general model. Sortie generation rates with and without dependent machine operations are shown comparatively with performance ranks in parenthesis. Note that overall performance declines significantly with dependent machine operations. This loss can be attributed to the idle time expended by ready aircraft that must wait for a formation companion. While the fully generalized manpower structure is optimal for each case, the performance ranks for the other structures vary. Dependent machine operations thus have a significant impact on the relative merit of the various manpower structures.

Table 4.3: Results with Dependent Machine Operations

g	x_1	x_2	x_3	x_4	x_5	R	R (dep.)
1	2	1	2	0	0	4.848 (4)	3.989 (2)
2	1	2	2	0	0	4.895 (2)	3.984 (3)
3	2	0	0	2	0	4.740 (5)	3.920 (5)
4	1	0	0	3	0	4.862 (3)	3.956 (4)
5	0	0	0	0	3	5.042 (1)	4.093 (1)

Chapter 5

Conclusions

In a survey of analytical methods for cyclic queues and closed queuing networks, a prominent theoretician observed that "despite the extent of their applications, the richness and explanatory power of these methods have not been appreciated or understood by many practitioners " [23, p. 605]. This thesis has demonstrated that the conceptual framework of a closed queuing network can offer valuable insight into a "real world" problem which practitioners have traditionally approached through other methods. Through specified modeling assumptions which enable the use of Markov decision analysis, a queuing network model can be employed to determine an optimal level of specialization and optimal task allocation for a maintenance manpower force. The method can be specifically applied to the problem of maximizing the operational effectiveness of a dispersed unit of military aircraft subject to a constraint on maintenance manpower expenditure.

Two important techniques can be applied to solve a queuing network model for manpower optimization: policy iteration and linear programming. The policy iteration approach allows an analyst to conjecture optimal task assignments for each manpower structure under consideration. If these conjectured assignments are good, a policy iteration algorithm will quickly converge to a solution. The policy iteration approach is particularly useful for problems where the number of candidate manpower structures is relatively small. The main disadvantage of this technique is that at least one set of simultaneous equations must be solved for every manpower structure under consideration.

Linear programming is another powerful tool which can be marshalled to find an optimal specialization strategy. A concurrent LP formulation can theoretically be developed to simultaneously consider all candidate manpower structures and assignment policies in a single model. The concurrent model is mathematically simple, but computational inefficiency and high storage requirements limit the utility of this approach for larger problems. These shortcomings can be avoided through an algorithm which sequentially solves models for each structure but eliminates much of the computational effort required for each solution. The sequential approach also provides full insight into the relative merits of all specialization strategies under consideration.

The queuing network approach to manpower optimization cannot completely displace all other evaluation methods. Some required assumptions may not be appropriate for a particular application. The extensions presented have demonstrated that creative modeling techniques can increase the flexibility of queuing network analysis, but the range of this flexibility is constrained. Another limitation is imposed by problem complexity. The queuing network model suffers from the "curse of dimensionality" [3, p. 323]. As the numbers of machines and task types increase, the system state space enlarges rapidly. Approximation methods which compromise the "discreteness" of machines in the system are not helpful because they mask an important characteristic of the problem. The use of approximate flow equivalence to reduce network size offers vital relief, but the practical applicability of the queuing network approach is still restricted to simple systems or highly aggregated analysis of complex systems.

Problem size, violations of assumptions, or the need to capture the effects of other dimensions of an overall logistics system may force a primary reliance on simulation modeling. However, a queuing network model may provide valuable insight

when used as an adjunct to simulation. For example, the analytical model can determine the assignment policies to be employed by candidate manpower structures in a simulation study. The simple "Mike's Flying Club" problem would require 78 simulation experiments to exhaustively evaluate all policies and structures, and multiple replications would be required to place statistical confidence bounds on each result. Preliminary queuing network analysis could instantly narrow the range of policies which merit investigation. The analytical model could also be employed to produce "external control variates" for increasing the computational efficiency of a simulation effort [25, p. 359]. Other adjunct uses are possible, such as using the analytical model to identify a starting point for a simulation search.

The principal value of the queuing network method lies is its ability to quickly reveal the relationships between key parameters, evaluate tradeoffs, offer fundamental insights, and answer basic questions. Listed below are some questions which might arise in the context of manpower analysis for military aircraft maintenance:

- How does a change in the manpower budget affect operational effectiveness? Is the optimal specialization strategy altered? How much effectiveness is lost if the budget declines but the specialization strategy remains unchanged? What effects result from changes in manpower training costs?

- What effect does changing the reliability or maintainability characteristics of a particular aircraft subsystem have on operational effectiveness? What is the impact of such a change on the optimal specialization strategy? How does the cost of improving the subsystem compare with the manpower costs that might be saved?

- What is the cost of dispersed operations? How much sortie generation capa-

bility is lost if aircraft are dispersed without an increase in total manpower? How much of an increase in manpower expenditure is required to maintain a specified sortie generation rate?

These questions are representative of myriad possible concerns. Similar issues might arise for many other enterprises which employ maintenance personnel of varying skills. These issues can be explored through the analytical method contributed by this thesis.

Appendix A

Policy Iteration Algorithm

This appendix includes a derivation and convergence proof for the policy iteration algorithm presented in Section 2.2. The algorithm can be described as a special case of the general policy iteration method developed by R. Howard [18].

Consider a machine maintenance system with Markovian transition rates r_{ij} defined as follows:

$$r_{ij} = \begin{cases} n_0^{(i)} q_{z^+(ij)} \mu_0 & \gamma(ij) = 0 \\ [s_{ij}/a_{\gamma(ij)}]\mu_{\gamma(ij)} & \gamma(ij) \in \{1, 2, \ldots, M\} \\ 0 & \text{otherwise} \end{cases}$$

During the time the system is in state i, it accumulates machine operating time at a rate $n_0^{(i)}$. Let dt be an infinitesimal time interval such that a system in state i will transition to state j in dt with probability $r_{ij}dt$. Let $\pi_j(t)$ be the probability that the system will be in state j at time t after the start of the Markov process. Further, let $v_i(t)$ be the total machine operating time that the system will achieve in time t if it is initialized in state i. Using these definitions, it follows that

$$\pi_j(t + dt) = \pi_j(t)\{1 - \sum_{i \neq j} r_{ji} dt\} + \sum_{i \neq j} \pi_i(t)\, r_{ij} dt \qquad j = 1, 2, \ldots, I \quad \text{(A.1)}$$

$$v_i(dt + t) = (1 - \sum_{j \neq i} r_{ij} dt)\{n_0^{(i)} dt + v_i(t)\} + \sum_{j \neq i} r_{ij} dt\, v_j(t) \quad i = 1, 2, \ldots, I \quad \text{(A.2)}$$

Equations A.1 and A.2 can be simplified by defining $r_{ii} = -\sum_{i \neq j} r_{ij}$ and substituting to yield

$$\begin{aligned} \pi_j(t+dt) &= \pi_j(t)\{1 + r_{jj}dt\} + \sum_{i \neq j} \pi_i(t) r_{ij} dt \\ &= \pi_j(t) + \sum_{i=1}^{I} \pi_i(t) r_{ij} dt \end{aligned} \tag{A.3}$$

$$\begin{aligned} v_i(t+dt) &= (1 + r_{ii}dt)\{n_0^{(i)}dt + v_i(t)\} + \sum_{j \neq i} r_{ij} v_j(t) dt \\ &= n_0^{(i)}dt + v_i(t) + n_0^{(i)} r_{ii} dt^2 + \sum_{j=1}^{I} r_{ij} v_j(t) dt \end{aligned} \tag{A.4}$$

Simple algebraic manipulation yields the expressions

$$\frac{\pi_j(t+dt) - \pi_j(t)}{dt} = \sum_{i=1}^{I} \pi_i(t) r_{ij} \tag{A.5}$$

$$\frac{v_i(t+dt) - v_i(t)}{dt} = n_0^{(i)} + n_0^{(i)} r_{ii} dt + \sum_{j=1}^{I} r_{ij} v_j(t) \tag{A.6}$$

Taking the limit $dt \rightarrow 0$ produces the constant-coefficient differential equations

$$\frac{d\pi_j(t)}{dt} = \sum_{i=1}^{I} \pi_i(t) r_{ij} \qquad j = 1, 2, \ldots, I \tag{A.7}$$

$$\frac{dv_i(t)}{dt} = n_0^{(i)} + \sum_{j=1}^{I} r_{ij} v_j(t) \qquad i = 1, 2, \ldots, I \tag{A.8}$$

These equations relate state probabilities to transition rates and govern the total expected machine operating time for the system in time t.

For all large t such that a Markov process reaches steady-state operation,

it is known that $d\pi_j(t)/dt = 0$ for each state j. Thus, Equations A.7 demand that

$$\sum_{i=1}^{I} \pi_i r_{ij} = 0 \qquad j = 1, 2, \ldots, I \tag{A.9}$$

where each π_i is a steady-state probability for all large t. Further, each $v_i(t)$ in each Equation A.8 can be replaced with an asymptotic expression $E(n_0)t + v_i$, where $E(n_0)$ is the slope of the asymptote (rate at which the system accumulates machine operating time) and v_i is the intercept. Equations A.8 then become

$$\begin{aligned} E(n_0) &= n_0^{(i)} + \sum_{j=1}^{I} r_{ij}\{E(n_0)t + v_j\} \\ &= n_0^{(i)} + E(n_0)t \sum_{j=1}^{I} r_{ij} + \sum_{j=1}^{I} r_{ij} v_j \end{aligned} \tag{A.10}$$

The expression $\sum_{j=1}^{I} r_{ij}$ is zero by definition, leaving the *value determination* equations

$$E(n_0) = n_0^{(i)} + \sum_{j=1}^{I} r_{ij} v_j \qquad i = 1, 2, \ldots, I \tag{A.11}$$

This set of I equations has $I + 1$ unknowns. Each v_i represents the relative value of beginning in a certain state, so any one of them can be set to an arbitrary value. For consistency, set $v_I = 0$.

Now consider two manpower assignment policies represented by decisions k' and k'' for each system state. Assume that the *policy improvement* step of the algorithm in Section 2.2 has produced each decision k'' as a successor to each decision k'. It is therefore known that

$$\sum_{j=1}^{I} r_{ij}^{k''} v_j' \geq \sum_{j=1}^{I} r_{ij}^{k'} v_j' \tag{A.12}$$

or

$$\delta_i = \sum_{j=1}^{I} r_{ij}^{k''} v_j' - \sum_{j=1}^{I} r_{ij}^{k'} v_j' \tag{A.13}$$

where $\delta_i \geq 0$. From the *value determination* Equations A.11, it is known that

$$E(n_0)'' = n_0^{(i)} + \sum_{j=1}^{I} r_{ij}^{k''} v_j'' \tag{A.14}$$

$$E(n_0)' = n_0^{(i)} + \sum_{j=1}^{I} r_{ij}^{k'} v_j' \tag{A.15}$$

Subtracting Equations A.15 from A.14 yields

$$\begin{aligned} E(n_0)'' - E(n_0)' &= \sum_{j=1}^{I} r_{ij}^{k''} v_j'' - \sum_{j=1}^{I} r_{ij}^{k'} v_j' \\ &= \sum_{j=1}^{I} r_{ij}^{k''} v_j'' + \delta_i - \sum_{j=1}^{I} r_{ij}^{k''} v_j' \\ &= \delta_i + \sum_{j=1}^{I} r_{ij}^{k''} (v_j'' - v_j') \end{aligned} \tag{A.16}$$

Let $E(n_0)^{\Delta} = E(n_0)'' - E(n_0)'$ and $v_i^{\Delta} = v_i'' - v_i'$. Equations A.16 then become

$$E(n_0)^{\Delta} = \delta_i + \sum_{j=1}^{I} r_{ij}^{k''} v_j^{\Delta} \qquad i = 1, 2, \ldots, I \tag{A.17}$$

Multiplying each Equation A.17 by a corresponding steady-state probability π_i'' and summing over all i yields

$$\begin{aligned} E(n_0)^{\Delta} &= \sum_{i=1}^{I} \delta_i \pi_i'' + \sum_{i=1}^{I} \pi_i'' \sum_{j=1}^{I} r_{ij}^{k''} v_j^{\Delta} \\ &= \sum_{i=1}^{I} \delta_i \pi_i'' + \sum_{j=1}^{I} v_j^{\Delta} \sum_{i=1}^{I} \pi_i'' r_{ij}^{k''} \end{aligned} \tag{A.18}$$

However, Equations A.9 require that $\sum_{i=1}^{I} \pi_i'' r_{ij}^{k''} = 0$ for each state j. Thus,

$$E(n_0)^{\Delta} = \sum_{i=1}^{I} \delta_i \pi_i'' \tag{A.19}$$

Since all $\pi_i'' \geq 0$ and all $\delta_i \geq 0$, it follows that $E(n_0)^{\Delta} \geq 0$. Thus, $E(n_0)''$ will be greater than $E(n_0)'$ if decisions k'' can be found such that $\sum_{j=1}^{I} r_{ij}^{k''} v_j > \sum_{j=1}^{I} r_{ij}^{k'} v_j$ for any state i.

It remains only to show that a better policy cannot exist without being found at some time by the *policy improvement* step. Assume that $E(n_0)'' > E(n_0)'$ but the algorithm has converged on the policy represented by decisions k'. Then $\delta_i \leq 0$ for all i when all δ_i are defined by Equations A.13. Since $\pi_i'' \geq 0$ for all i, Equation A.19 demands that $E(n_0)'' - E(n_0)' \leq 0$. However, this contradicts the assumption that $E(n_0)'' > E(n_0)'$. It is therefore impossible for a superior policy to exist but remain undiscovered at the completion of the algorithm.

Appendix B

Program MAINTOP

B.1 Description

Maintenance resource optimization using the sequential linear programming approach is implemented in a portable computer program called MAINTOP (MAINTenance OPtimization). The program is written in standard Pascal and is listed in the next section of this appendix. The listing includes file statements for both mainframe and microcomputer implementations. Some internal documentation is provided, including variable definitions and brief descriptions for each subroutine. An overview of the relationship between the various program components can be obtained from Figure B.1.

Program MAINTOP makes extensive use of recursive calls to subroutines. This approach achieves programming efficiency in constructing the queuing network, building the system state space, defining alternative resource structures, and identifying nondominated feasible decisions for resource assignments. The program also employs pointer variables and linked lists to preserve computer memory and thus increase the tractable problem size. Maximum use of computer capability can be achieved by adjusting the constant parameters listed at the beginning of the program. During program execution, network reduction will be performed as necessary to limit the size of the system state space.

To use program MAINTOP, data should first be entered into a text file called `MAINTOP.DAT` in accordance with the format displayed in Table B.1. The

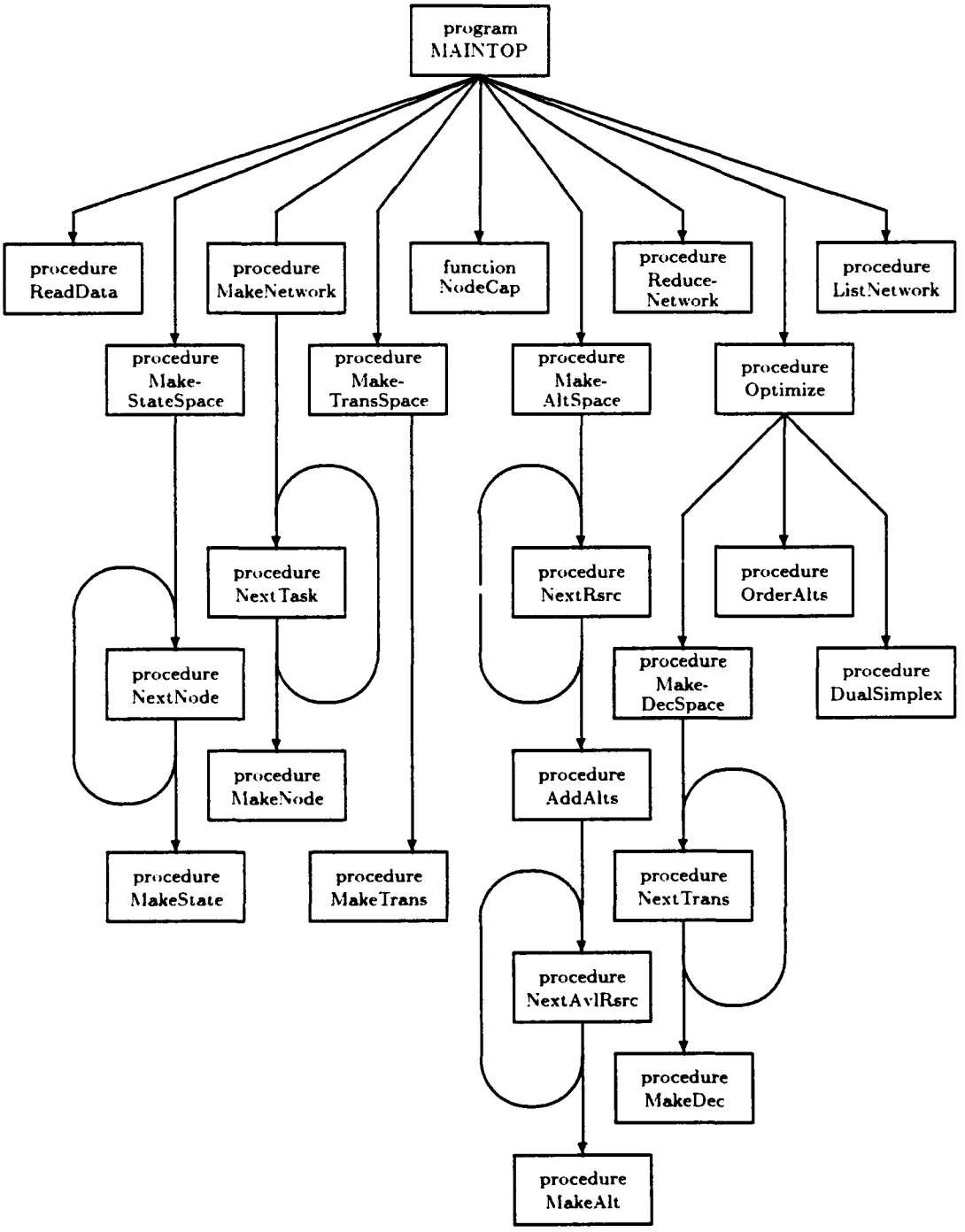


Figure B.1: Relationship between Components of Program MAINTOP

general format is paralleled by a sample data file for the "Mike's Flying Club" problem introduced in Chapter 2. Precise character position is not critical, provided the data is sequenced properly and placed in separate lines for each maintenance task and resource. For each task m, the set Φ_m is defined as the set of all tasks which must be completed before task m can be initiated. A value of 0 in the λ_m position indicates that a task is always required between operational activities. A value of 0 is also used as an "end of list" character for the task and resource parameters. Care should be taken to avoid trailing blanks on any line of data.

Table B.1: Data Format for Program MAINTOP

N	μ_0	C			
:					
m	μ_m	a_m	λ_m	Φ_m	
:					
0					
:					
y	c_y	Q_y			
:					
0					

2	0.50	100.0		
1	1.00	1	0	2 3
2	0.25	1	0.20	
3	0.50	2	0.25	
0				
1	10.00	1		
2	20.00	2		
3	25.00	3		
4	30.00	2 3		
5	33.00	1 2 3		
0				

Output from the program is recorded to a text file called `MAINTOP.OUT`. The output includes an echo listing of input parameters, a description of the generated model, and a presentation of performance results. Sample output for the "Mike's Flying Club" problem is listed in the final section of this appendix.

B.2 Program Listing

```
program MAINTOP(input,output,Data,Out);

{*************************************************************************}
{*                                                                       *}
{*    Determines optimal specialization and task allocation for maintenance  *}
{*    resources using a queuing network formulation.                     *}
{*                                                                       *}
{*    Authors:   Dennis C. Dietz and Matthew Rosenshine                  *}
{*               Department of Industrial and Management Systems Engineering  *}
{*               Pennsylvania State University, University Park, PA  16802    *}
{*                                                                       *}
{*    Language:  Standard Pascal -- program is suitable for mainframe or     *}
{*               microcomputer implementation.                           *}
{*                                                                       *}
{*************************************************************************}

const MaxTask=10;    {maximum number of task types}
      MaxRsrc=25;    {maximum number of resource types}
      MaxSpec=100;   {maximum number of specialization strategies}
      MaxNode=250;   {maximum number of network nodes}
      MaxState=500;  {maximum number of system states}
      MaxDec=2000;   {maximum number of decision variables}

type TaskSet = set of 1..MaxTask;
     NodeSet = set of 0..MaxNode;
     RsrcSet = set of 1..MaxRsrc;
     TaskType = record
         TaskRate: real;     {rate of task completion}
         ReqRsrc: integer;   {number of maintenance resources required}
         FailRate: real;     {rate of random failure}
         Precede: TaskSet    {set of tasks this task must precede}
       end; {TaskType}
     TaskArray = array [1..MaxTask] of TaskType;
     TaskIntArray = array [1..MaxTask] of integer;
     TaskTstArray = array [1..MaxTask] of boolean;
     RsrcType = record
         Cost: real;          {unit cost of resource}
         TaskQual: TaskSet;   {set of tasks this resource can perform}
         MinRsrc: integer;    {minimum resource availability}
         MaxRsrc: integer;    {maximum resource availability}
         StpRsrc: integer     {availability increase increment}
       end; {RsrcType}
     RsrcArray = array [1..MaxRsrc] of RsrcType;
     RsrcIntArray = array [1..MaxRsrc] of integer;
     NodeType = record
         Prob: real;          {routing probability}
         RedAdj: real;        {rate adjustment for network reduction}
         Pending: TaskSet;    {set of pending tasks}
```

```
        Eligible: TaskSet;  {set of eligible tasks}
        NextNodes: NodeSet  {set of succeeding nodes}
      end; {NodeType}
    NodeArray = array [0..MaxNode] of NodeType;
    NodeIntArray = array [0..MaxNode] of integer;
    OccupyPointer = ^OccupyType;
    OccupyType = record
        Node: 0..MaxNode;          {occupied node identification}
        Machs: integer;            {number of machines at node}
        NextOccupy: OccupyPointer  {pointer to next occupied node}
      end; {OccupyType}
    StateArray = array [1..MaxState] of OccupyPointer;
    StateIntArray = array [1..MaxState] of integer;
    StateRealArray = array [1..MaxState] of real;
    TransPointer = ^TransType;
    TransType = record
        NextState: 1..MaxState;  {new state after transition}
        LossNode: 0..MaxNode;    {node which loses a machine}
        GainNode: 0..MaxNode;    {node which gains a machine}
        TransTask: 1..MaxTask;   {task completed at transition}
        NextTrans: TransPointer  {pointer to next transition}
      end; {TransType}
    TransArray = array [1..MaxState] of TransPointer;
    DecMatrix = array [0..MaxState,0..MaxDec] of real;
    ExPointer = ^ExType;
    ExType = record
        ExDec: 1..MaxDec;  {excluded decision}
        NextEx: ExPointer  {pointer to next excluded decision}
      end; {ExType}
    AltPointer = ^AltType;
    AltType = record
        Structure: RsrcIntArray;     {resource structure}
        SpecCode: 1..MaxSpec;        {specialization strategy}
        Excluded: ExPointer;         {pointer to excluded decisions}
        CrntEx: ExPointer;           {pointer to current excluded decision}
        Accept: boolean;             {indicator for acceptable structure}
        GoodGuess: real;             {estimater for relative performance}
        Optimum: real;               {expected number of operating machines}
        NextAlt,PrevAlt: AltPointer  {pointer to next/previous alternative}
      end; {AltType}
    OptArray = array [1..MaxSpec] of AltPointer;

var NumMach: integer;          {number of machines in system}
    NumTask: integer;          {number of task types}
    NumRsrc: integer;          {number of resource specialty types}
    NumNode: integer;          {number of nodes in network}
    NumState: integer;         {number of system states}
    NumDec: integer;           {number of decision variables}
    NodeLimit: integer;        {number of nodes in reduced network}
    OpsRate: real;             {rate of operational activity completion}
    CostLimit: real;           {resource expenditure limit}
```

```
    Method: char;              {solution method}
    TaskData: TaskArray;       {array of task data}
    LowNode: TaskIntArray;     {array of low number nodes for task set size}
    RsrcData: RsrcArray;       {array of resource data}
    NodeData: NodeArray;       {array of node data}
    StateData: StateArray;     {array of state data}
    TransData: TransArray;     {array of transition data}
    AltData: AltPointer;       {pointer to list of structure alternatives}
    Dec: DecMatrix;            {matrix of decision coefficients}
    BasicDec: StateIntArray;   {array of basic decisions}
    Data,Out: text;            {input and output text files}

{*****************************************************************************}

procedure ReadData(var NumMach,NumTask,NumRsrc: integer;
                   var OpsRate,CostLimit: real;
                   var Method: char;
                   var TaskData: TaskArray;
                   var RsrcData: RsrcArray);

{-----------------------------------------------------------------------------}
{   Reads model parameters from input data file and echos to output.          }
{                                                                            }
{   Calling Routines:  program MAINTOP                                       }
{   Called Routines:   none                                                  }
{-----------------------------------------------------------------------------}

var I,J: integer;  {local counter variables}

begin
    writeln('** Reading Data ...');
    writeln(Out,'**** INPUT DATA SUMMARY ****');
    writeln(Out);
    readln(Data,NumMach,OpsRate,CostLimit);
    writeln(Out,'Number of Machines: ',NumMach:8);
    writeln(Out,'Operations Rate:    ',OpsRate:8:4);
    writeln(Out,'Expenditure Limit:  ',CostLimit:8:2);
    write(Out,'Solution Method:    ');
    writeln;
    writeln('--- Solution Method Menu ---');
    writeln('    N.  Optimize each specialization strategy (NORMAL)');
    writeln('    I.  Evaluate all resource structures (INSIGHT)');
    writeln('    S.  Find optimal structure only (SPEED)');
    writeln;
    Method:=' ';
    repeat
        write('Select solution method -->  ');
        readln(Method);
        writeln
      until (Method in ['N','n','I','i','S','s']);
    writeln;
```

```
case Method of
    'N','n': writeln(Out,'  NORMAL');
    'I','i': writeln(Out,' INSIGHT');
    'S','s': writeln(Out,'   SPEED')
  end; {case}
writeln(Out);
writeln(Out);
NumTask:=0;
for I:=1 to MaxTask do TaskData[I].Precede:=[];
read(Data,I);
while (I<>0) do begin
    writeln(Out,'Task ',I:2);
    NumTask:=NumTask+1;
    with TaskData[NumTask] do begin
        read(Data,TaskRate,ReqRsrc,FailRate);
        writeln(Out,'Repair Rate:        ',TaskRate:8:4);
        writeln(Out,'Resources per Task: ',ReqRsrc:8);
        write(Out,  'Failure Rate:       ');
        if (FailRate>0.0) then writeln(Out,FailRate:8:4)
          else writeln(Out,'      --')
      end; {with}
    write(Out,'Prerequisite Tasks: ');
    while not eoln(Data) do begin
        read(Data,J);
        write(Out,J:4);
        with TaskData[J] do Precede:=Precede+[I]
      end; {while}
    writeln(Out);
    writeln(Out);
    read(Data,I)
  end; {while}
writeln(Out);
NumRsrc:=0;
read(Data,I);
while (I<>0) do begin
    writeln(Out,'Resource ',I:2);
    NumRsrc:=NumRsrc+1;
    with RsrcData[NumRsrc] do begin
        read(Data,Cost);
        writeln(Out,'Unit Cost:   ',Cost:8:2);
        TaskQual:=[];
        write(Out,'Qualified Tasks: ');
        while not eoln(Data) do begin
            read(Data,J);
            write(Out,J:4);
            TaskQual:=TaskQual+[J]
          end; {while}
      end; {with}
    writeln(Out);
    writeln(Out);
    read(Data,I)
```

```
      end; {while}
    writeln(Out);
  end; {ReadData}

{*****************************************************************************}

procedure MakeNode(var TaskSetSz,NumTask,NumNode: integer;
                   var OpsRate,RateSum: real;
                   var TaskData: TaskArray;
                   var LowNode: TaskIntArray;
                   var NodeData: NodeArray;
                   var TmpTaskSet,PrecedeSet: TaskSet);

{-----------------------------------------------------------------------------}
{   Creates a network node and computes node parameters.                       }
{                                                                              }
{   Calling Routines:  procedure NextTask                                      }
{   Called Routines:   none                                                    }
{-----------------------------------------------------------------------------}

var I,J: integer;  {local counter variables}

begin
    NumNode:=NumNode+1;
    if (LowNode[TaskSetSz]=0) then LowNode[TaskSetSz]:=NumNode;
    with NodeData[NumNode] do begin
        Pending:=TmpTaskSet;
        Eligible:=TmpTaskSet-PrecedeSet;
        Prob:=OpsRate/RateSum;
        for I:=1 to NumTask do
          if (TaskData[I].FailRate=0.0) and not (I in Pending) then Prob:=0.0;
        if (Prob>0.0) then
          for I:=1 to NumNode-1 do
            if (NodeData[I].Pending<=TmpTaskSet) then
              Prob:=Prob-NodeData[I].Prob;
        RedAdj:=1.0;
        if (TaskSetSz=1) then NextNodes:=[0]
          else begin
              NextNodes:=[];
              for I:=LowNode[TaskSetSz-1] to LowNode[TaskSetSz]-1 do
                for J:=1 to NumTask do
                  if (J in NodeData[NumNode].Eligible)
                    and (NodeData[I].Pending+[J]=TmpTaskSet) then
                      NextNodes:=NextNodes+[I]
            end; {else}
      end; {with}
  end; {MakeNode}

{*****************************************************************************}

procedure NextTask(I,TaskSetSz: integer;
```

```
                    var HiFirstTask,TaskCnt: integer;
                    var NumTask,NumNode: integer;
                    var OpsRate: real;
                    var TaskData: TaskArray;
                    var LowNode: TaskIntArray;
                    var NodeData: NodeArray;
                    var TmpTaskSet: TaskSet);

{-------------------------------------------------------------------------------}
{   Recursively builds a pending task set and determines if a node with the    }
{   current pending task set should be created.                                }
{                                                                              }
{   Calling Routines:  procedure MakeNetwork, procedure NextTask               }
{   Called Routines:   procedure MakeNode                                      }
{-------------------------------------------------------------------------------}

var J: integer;           {local counter variable}
    RateSum: real;        {sum of eligible task completion rates}
    PrecedeSet: TaskSet;  {tasks which must precede all pending tasks}
    AlwaysSet: TaskSet;   {pending tasks which are always required}

begin
    if (NumNode<MaxNode) then begin
        TmpTaskSet:=TmpTaskSet+[I];
        TaskCnt:=TaskCnt+1;
        if (TaskCnt=TaskSetSz) then begin
            PrecedeSet:=[];
            AlwaysSet:=[];
            RateSum:=OpsRate;
            for J:=1 to NumTask do
              if (J in TmpTaskSet) then
                PrecedeSet:=PrecedeSet+TaskData[J].Precede
                  else RateSum:=RateSum+TaskData[J].FailRate;
            for J:=1 to NumTask do
              if (J in PrecedeSet) and (TaskData[J].FailRate=0.0) then
                AlwaysSet:=AlwaysSet+[J];
            if (AlwaysSet<=TmpTaskSet) then
              MakeNode(TaskSetSz,NumTask,NumNode,OpsRate,RateSum,TaskData,
                LowNode,NodeData,TmpTaskSet,PrecedeSet)
          end else begin
            HiFirstTask:=HiFirstTask+1;
            for J:=I+1 to HiFirstTask do
              NextTask(J,TaskSetSz,HiFirstTask,TaskCnt,NumTask,NumNode,
                OpsRate,TaskData,LowNode,NodeData,TmpTaskSet);
            HiFirstTask:=HiFirstTask-1
          end; {else}
        TmpTaskSet:=TmpTaskSet-[I];
        TaskCnt:=TaskCnt-1
      end;
  end; {NextTask}
```

```
{*****************************************************************************}

procedure MakeNetwork(var NumTask,NumNode: integer;
                      var OpsRate: real;
                      var TaskData: TaskArray;
                      var LowNode: TaskIntArray;
                      var NodeData: NodeArray);

{---------------------------------------------------------------------------}
{   Creates a queuing network from model parameters.                        }
{                                                                           }
{   Calling Routines:  program MAINTOP                                      }
{   Called Routines:   procedure NextTask                                   }
{---------------------------------------------------------------------------}


var I: integer;             {local counter variable}
    TmpTaskSet: TaskSet;    {temporary set of pending tasks}
    HiFirstTask: integer;   {highest numbered first task in set}
    TaskSetSz: integer;     {number of tasks in set}
    TaskCnt: integer;       {task counter}

begin
    writeln('** Building Network ...');
    for I:=1 to NumTask do LowNode[I]:=0;
    NumNode:=0;
    TmpTaskSet:=[];
    TaskCnt:=0;
    for TaskSetSz:=1 to NumTask do begin
        HiFirstTask:=NumTask-TaskSetSz+1;
        for I:=1 to HiFirstTask do
          NextTask(I,TaskSetSz,HiFirstTask,TaskCnt,NumTask,NumNode,
            OpsRate,TaskData,LowNode,NodeData,TmpTaskSet)
      end; {for}
  end; {MakeNetwork}

{*****************************************************************************}

function NodeCap(var NumMach,NumNode: integer): integer;

{---------------------------------------------------------------------------}
{   Returns the maximum number of nodes permitted by the maximum system     }
{   state size (MaxState).                                                  }
{                                                                           }
{   Calling Routines:  program MAINTOP                                      }
{   Called Routines:   none                                                 }
{---------------------------------------------------------------------------}

var I: integer;           {local counter variable}
    StSpaceSz: integer;   {state space size}
```

```
begin
    StSpaceSz:=NumMach+1;
    I:=1;
    while (I<=NumNode) and (StSpaceSz<MaxState) do begin
        I:=I+1;
        StSpaceSz:=StSpaceSz*(NumMach+I) div I
      end; {while}
    NodeCap:=I-1
  end; {NodeCap}

{*****************************************************************************}

procedure ReduceNetwork(var NumTask,NumNode,NodeLimit: integer;
                        var TaskData: TaskArray;
                        var LowNode: TaskIntArray;
                        var NodeData: NodeArray);

{-----------------------------------------------------------------------------}
{   Reduces network size and adjusts node parameters.                         }
{                                                                             }
{   Calling Routines:  program MAINTOP                                        }
{   Called Routines:   none                                                   }
{-------------------------------------------------------------------------------}

var I,J,K: integer;       {local counter variables}
    TaskSetSz: integer;   {number of tasks in set}
    RateI: real;          {departure rate from losing node}
    RateJ: real;          {departure rate from gaining node}
    ProbIJ: real;         {transition probability from losing to gaining node}
    NewProbJ: real;       {adjusted routing probability}
    NewRateJ: real;       {adjusted departure rate}

begin
    writeln('** Reducing Network ...');
    for I:=NumNode downto NodeLimit+1 do begin
        TaskSetSz:=0;
        RateI:=0.0;
        for K:=1 to NumTask do begin
            if (K in NodeData[I].Pending) then TaskSetSz:=TaskSetSz+1;
            if (K in NodeData[I].Eligible) then
              RateI:=RateI+TaskData[K].TaskRate
          end; {for}
        for J:=LowNode[TaskSetSz-1] to LowNode[TaskSetSz]-1 do
          if (J in NodeData[I].NextNodes) then begin
              RateJ:=0.0;
              for K:=1 to NumTask do begin
                  if (K in NodeData[J].Eligible) then
                    RateJ:=RateJ+TaskData[K].TaskRate;
                  if (NodeData[J].Eligible+[K]=NodeData[I].Eligible) then
                    ProbIJ:=TaskData[K].TaskRate/RateI
                end; {for}
```

```
            NewProbJ:=NodeData[J].Prob+NodeData[I].Prob*ProbIJ;
            NewRateJ:=NewProbJ/(NodeData[J].Prob/(RateJ*NodeData[J].RedAdj)
              +NodeData[I].Prob*ProbIJ*(1.0/(RateJ*NodeData[J].RedAdj)
                +1.0/(RateI*NodeData[I].RedAdj)));
            NodeData[J].Prob:=NewProbJ;
            NodeData[J].RedAdj:=NewRateJ/(RateJ*NodeData[J].RedAdj)
          end; {if}
      end; {for}
  end; {ReduceNetwork}

{*****************************************************************************}

procedure ListNetwork(var NumTask,NumNode,NodeLimit: integer;
                      var NodeData: NodeArray);

{-----------------------------------------------------------------------------}
{   Writes network parameters to output file.                                 }
{                                                                             }
{   Calling Routines:  program MAINTOP                                        }
{   Called Routines:   none                                                   }
{-----------------------------------------------------------------------------}

var I,J: integer;  {local counter variables}

begin
    writeln(Out,'**** MODEL SUMMARY ****');
    writeln(Out);
    writeln(Out,'Number of maintenance nodes in full network: ',NumNode:4);
    if (NodeLimit<NumNode) then
      writeln(Out,'Number of maintenance nodes in reduced network: ',
        NodeLimit:4);
    if (NodeLimit<NumTask) then
      writeln(Out,'WARNING:  Fewer allowable nodes than tasks.');
    writeln(Out);
    for I:=1 to NodeLimit do begin
        writeln(Out,'Node ',I:4);
        with NodeData[I] do begin
            writeln(Out,'Routing Probability:   ',Prob:5:4);
            if (NodeLimit<NumNode) then
              writeln(Out,'Reduction Adjustment:  ',RedAdj:5:4);
            write(Out,'Pending Tasks:  ');
            for J:=1 to NumTask do
              if (J in Pending) then write(Out,J:4);
            writeln(Out);
            write(Out,'Eligible Tasks: ');
            for J:=1 to NumTask do
              if (J in Eligible) then write(Out,J:4);
            writeln(Out)
          end; {with}
        writeln(Out)
      end; {for}
```

```
  end; {ListNetwork}

{*****************************************************************************}

procedure MakeState(var NodeLimit,NumState: integer;
                    var StateData: StateArray;
                    var TmpState: NodeIntArray);

{---------------------------------------------------------------------------}
{   Creates a system state.  The machine distribution is stored in a linked   }
{   list of occupied nodes.                                                   }
{                                                                             }
{   Calling Routines:  procedure NextTrans                                    }
{   Called Routines:   none                                                   }
{---------------------------------------------------------------------------}

var I: integer;                 {local counter variable}
    CrntOccupy: OccupyPointer;  {pointer to current occupied node}

begin
    NumState:=NumState+1;
    for I:=0 to NodeLimit do
      if (TmpState[I]>0) then begin
          if (StateData[NumState]=nil) then begin
              new(StateData[NumState]);
              CrntOccupy:=StateData[NumState]
            end else begin
              new(CrntOccupy^.NextOccupy);
              CrntOccupy:=CrntOccupy^.NextOccupy
            end; {else}
          with CrntOccupy^ do begin
              Node:=I;
              Machs:=TmpState[I];
              NextOccupy:=nil
            end; {with}
        end; {if}
  end; {MakeState}

{*****************************************************************************}

procedure NextNode(I: integer;
                   var NodeCnt,NodeLimit,NumState: integer;
                   var StateData: StateArray;
                   var TmpState: NodeIntArray);

{---------------------------------------------------------------------------}
{   Recursively distributes machines to network nodes and identifies system   }
{   states.                                                                   }
{                                                                             }
{   Calling Routines:  procedure MakeStateSpace, procedure NextNode           }
{   Called Routines:   procedure MakeState                                    }
```

```
{-------------------------------------------------------------------------}

var J: integer;  {local counter variable}

begin
    NodeCnt:=NodeCnt+1;
    if (NodeCnt=NodeLimit) then begin
        TmpState[NodeCnt]:=I;
        MakeState(NodeLimit,NumState,StateData,TmpState)
      end else
        for J:=0 to I do begin
            TmpState[NodeCnt]:=I-J;
            NextNode(J,NodeCnt,NodeLimit,NumState,StateData,TmpState)
          end; {for}
    NodeCnt:=NodeCnt-1
  end; {NextNode}

{*************************************************************************}

procedure MakeStateSpace(var NumMach,NodeLimit,NumState: integer;
                         var StateData: StateArray);

{-------------------------------------------------------------------------}
{   Creates the system state space.                                       }
{                                                                         }
{   Calling Routines:  program MAINTOP                                    }
{   Called Routines:   procedure NextNode                                 }
{-------------------------------------------------------------------------}

var I: integer;              {local counter variable}
    NodeCnt: integer;        {node counter}
    TmpState: NodeIntArray;  {temporary state data}

begin
    writeln('** Defining System States ...');
    for I:=1 to MaxState do StateData[I]:=nil;
    NumState:=0;
    NodeCnt:=0;
    for I:=0 to NumMach do begin
        TmpState[NodeCnt]:=NumMach-I;
        NextNode(I,NodeCnt,NodeLimit,NumState,StateData,TmpState)
      end; {for}
    writeln(Out,'Number of system states: ',NumState:6);
    writeln(Out)
  end; {MakeStateSpace}

{*************************************************************************}

procedure MakeTrans(I,J: integer;
                    var Loser,Gainer,NumTask: integer;
                    var TaskData: TaskArray;
```

```
                    var NodeData: NodeArray;
                    var TransData: TransArray;
                    var CrntTrans: TransPointer);

{-------------------------------------------------------------------------------}
{   Creates a data record for transition parameters.                            }
{                                                                               }
{   Calling Routines:  procedure MakeTransSpace                                 }
{   Called Routines:   none                                                     }
{-------------------------------------------------------------------------------}

var K: integer;           {local counter variable}
    TmpTaskSet: TaskSet;  {temporary set of pending tasks}

begin
    if (TransData[I]=nil) then begin
        new(TransData[I]);
        CrntTrans:=TransData[I]
      end else begin
        new(CrntTrans^.NextTrans);
        CrntTrans:=CrntTrans^.NextTrans
      end; {else}
    with CrntTrans^ do begin
        NextState:=J;
        if (Loser=0) then begin
            LossNode:=0;
            GainNode:=Gainer;
            TransTask:=0
          end else begin
            LossNode:=Loser;
            GainNode:=Gainer;
            if (Gainer=0) then TmpTaskSet:=[]
              else TmpTaskSet:=NodeData[Gainer].Pending;
            for K:=1 to NumTask do
              if ([K]+TmpTaskSet=NodeData[Loser].Pending) then TransTask:=K
          end; {else}
        NextTrans:=nil
      end; {with}
  end; {MakeTrans}

{*******************************************************************************}

procedure MakeTransSpace(var NumTask,NumState,NodeLimit: integer;
                         var NodeData: NodeArray;
                         var StateData: StateArray;
                         var TransData: TransArray);

{-------------------------------------------------------------------------------}
{   Creates a table of transition data.  For each state, transition data is     }
{   stored in a linked list of records to save memory.                          }
{                                                                               }
```

```
{   Calling Routines:  program MAINTOP                                          }
{   Called Routines:   none                                                     }
{-------------------------------------------------------------------------------}

var I,J,K: integer;                      {local counter variables}
    ChgMach: integer;                    {change in machines at a node}
    Loser,Gainer: integer;               {losing and gaining nodes}
    TmpStateI,TmpStateJ: NodeIntArray;   {temporary state data}
    CrntI,CrntJ: OccupyPointer;          {pointers to current state data}
    CrntTrans: TransPointer;             {pointer to current transition}
    OneStep: boolean;                    {indicator for one-step behavior}

begin
    writeln('** Defining Transitions ...');
    for I:=1 to NumState do begin
        TransData[I]:=nil;
        for K:=0 to Nodelimit do TmpStateI[K]:=0;
        CrntI:=StateData[I];
        while (CrntI<>nil) do begin
            TmpStateI[CrntI^.Node]:=CrntI^.Machs;
            CrntI:=CrntI^.NextOccupy
          end; {while}
        for J:=1 to NumState do if (I<>J) then begin
            for K:=0 to NodeLimit do TmpStateJ[K]:=0;
            CrntJ:=StateData[J];
            while (CrntJ<>nil) do begin
                TmpStateJ[CrntJ^.Node]:=CrntJ^.Machs;
                CrntJ:=CrntJ^.NextOccupy
              end; {while}
            OneStep:=true;
            Loser:=-1;
            Gainer:=-1;
            K:=0;
            while (OneStep=true) and (K<=NodeLimit) do begin
                ChgMach:=TmpStateJ[K]-TmpStateI[K];
                if (abs(ChgMach)>1) then OneStep:=false
                  else case ChgMach of
                     -1: if (Loser>=0) then OneStep:=false else Loser:=K;
                      1: if (Gainer>=0) then OneStep:=false else Gainer:=K
                    end; {case}
                 K:=K+1
              end; {while}
            if (OneStep=true) and
              ((Loser=0) or (Gainer in NodeData[Loser].NextNodes)) then
                MakeTrans(I,J,Loser,Gainer,NumTask,TaskData,NodeData,TransData,
                  CrntTrans)
          end; {for}
      end; {for}
  end; {MakeTransSpace}

{*******************************************************************************}
```

```
procedure MakeAlt(var SpecCnt: integer;
                  var AltData,CrntAlt: AltPointer;
                  var TmpStructure: RsrcIntArray);

{-------------------------------------------------------------------------------}
{   Creates a maintenance resource structure alternative.                       }
{                                                                               }
{   Calling Routines:  procedure NextAvlRsrc                                    }
{   Called Routines:   none                                                     }
{-------------------------------------------------------------------------------}

begin
    if (AltData=nil) then begin
        new(AltData);
        AltData^.PrevAlt:=nil;
        CrntAlt:=AltData
      end else begin
        new(CrntAlt^.NextAlt);
        CrntAlt^.NextAlt^.PrevAlt:=CrntAlt;
        CrntAlt:=CrntAlt^.NextAlt
      end; {else}
    with CrntAlt^ do begin
        Structure:=TmpStructure;
        SpecCode:=SpecCnt;
        Excluded:=nil;
        CrntEx:=nil;
        NextAlt:=nil
      end; {with}
  end; {MakeAlt}

{*******************************************************************************}

procedure NextAvlRsrc(I: integer;
                      CostRem: real;
                      var SpecCnt,NumRsrc: integer;
                      var RsrcData: RsrcArray;
                      var AltData,CrntAlt: AltPointer;
                      var TmpStructure: RsrcIntArray);

{-------------------------------------------------------------------------------}
{   Recursively adds available maintenance resources to create structure        }
{   alternatives.                                                               }
{                                                                               }
{   Calling Routines:  procedure AddAlts, procedure NextAvlRsrc                 }
{   Called Routines:   procedure MakeAlt                                        }
{-------------------------------------------------------------------------------}

var J: integer;        {local counter variable}
var Accept: boolean;   {indicator for acceptable alternative}
```

```
begin
    while (I<=NumRsrc) and (TmpStructure[I]=0) do I:=I+1;
    if (I>NumRsrc) then begin
        if (CostRem>=0.0) then begin
            Accept:=true;
            for J:=1 to NumRsrc do
              with RsrcData[J] do
                if (TmpStructure[J]>0) and (MaxRsrc-TmpStructure[J]>=StpRsrc)
                  and (Cost*StpRsrc<=CostRem) then Accept:=false;
            if Accept then MakeAlt(SpecCnt,AltData,CrntAlt,TmpStructure)
          end; {if}
      end else
        with RsrcData[I] do begin
            J:=MinRsrc;
            while (J<=MaxRsrc) do begin
                if (CostRem>=0.0) then begin
                    TmpStructure[I]:=J;
                    NextAvlRsrc(I+1,CostRem,SpecCnt,NumRsrc,RsrcData,
                      AltData,CrntAlt,TmpStructure)
                  end; {if}
                J:=J+StpRsrc;
                CostRem:=CostRem-StpRsrc*Cost
              end; {while}
          end; {with}
  end; {NextAvlRsrc}

{*****************************************************************************}

procedure AddAlts(var SpecCnt,NumMach,NumTask,NodeLimit,NumRsrc: integer;
                  var CostLimit: real;
                  var TaskData: TaskArray;
                  var NodeData: NodeArray;
                  var RsrcData: RsrcArray;
                  var AltData,CrntAlt: AltPointer;
                  var TmpStructure: RsrcIntArray);

{-----------------------------------------------------------------------------}
{   Adds resource structure alternatives to a linked list.                    }
{                                                                             }
{   Calling Routines:  procedure NextRsrc                                     }
{   Called Routines:   procedure NextAvlRsrc                                  }
{-----------------------------------------------------------------------------}

var I,J,K: integer;    {local counter variables}
    NodeReq: integer;  {number of resources required at node}
    CostRem: real;     {remaining available expenditure}

begin
    CostRem:=CostLimit;
    SpecCnt:=SpecCnt+1;
    write(Out,SpecCnt:8,'         { ');
```

```
    for I:=1 to NumRsrc do begin
        if (TmpStructure[I]>0) then begin
            write(Out,I:1,' ');
            with RsrcData[I] do begin
                MinRsrc:=0;
                MaxRsrc:=0;
                StpRsrc:=9999;
                for J:=1 to NumTask do
                  if (J in TaskQual) then begin
                      if (TaskData[J].ReqRsrc>MinRsrc) then
                        MinRsrc:=TaskData[J].ReqRsrc;
                      if (TaskData[J].ReqRsrc<StpRsrc) then
                        StpRsrc:=TaskData[J].ReqRsrc
                    end; {if}
                for J:=1 to NodeLimit do begin
                    NodeReq:=0;
                    for K:=1 to NumTask do begin
                        if (K in TaskQual) and (K in NodeData[J].Eligible) then
                          NodeReq:=NodeReq+TaskData[K].ReqRsrc*NumMach;
                        if (NodeReq>MaxRsrc) then MaxRsrc:=NodeReq
                      end; {for}
                  end; {for}
                CostRem:=CostRem-MinRsrc*Cost
              end; {with}
          end; {if}
      end; {for}
    writeln(Out,'}');
    I:=1;
    NextAvlRsrc(I,CostRem,SpecCnt,NumRsrc,RsrcData,AltData,CrntAlt,TmpStructure)
  end; {AddAlts}

{*****************************************************************************}

procedure NextRsrc(I: integer;
                   TaskCover: TaskTstArray;
                   var SpecCnt,NumMach,NumTask,NodeLimit,NumRsrc: integer;
                   var CostLimit: real;
                   var TaskData: TaskArray;
                   var NodeData: NodeArray;
                   var RsrcData: RsrcArray;
                   var AltData,CrntAlt: AltPointer;
                   var TmpStructure: RsrcIntArray);

{-----------------------------------------------------------------------------}
{   Recursively determines acceptable resource specialization strategies.     }
{   Adds alternative resource structures whenever an acceptable strategy      }
{   is found.                                                                 }
{                                                                             }
{   Calling Routines:  procedure MakeAltSpace, procedure NextRsrc             }
{   Called Routines:   procedure AddAlts                                      }
{-----------------------------------------------------------------------------}
```

```
var J: integer;        {local counter variable}
    Cont: boolean;     {indicator for continued addition of resource types}
    Accept: boolean;   {indicator for acceptable strategy}

begin
    TmpStructure[I]:=1;
    Cont:=true;
    for J:=1 to NumTask do
      if (J in RsrcData[I].TaskQual) then begin
          if TaskCover[J] then Cont:=false else TaskCover[J]:=true
        end; {if}
    if Cont then begin
        Accept:=true;
        for J:=1 to NumTask do
          if not TaskCover[J] then Accept:=false;
        if Accept then
            AddAlts(SpecCnt,NumMach,NumTask,NodeLimit,NumRsrc,CostLimit,
              TaskData,NodeData,RsrcData,AltData,CrntAlt,TmpStructure)
            else for J:=I+1 to NumRsrc do
              NextRsrc(J,TaskCover,SpecCnt,NumMach,NumTask,NodeLimit,NumRsrc,
                CostLimit,TaskData,NodeData,RsrcData,AltData,CrntAlt,
                  TmpStructure);
      end; {if}
    TmpStructure[I]:=0;
  end; {NextRsrc}

{*****************************************************************************}

procedure MakeAltSpace(var NumMach,NumTask,NodeLimit,NumRsrc: integer;
                       var CostLimit: real;
                       var TaskData: TaskArray;
                       var NodeData: NodeArray;
                       var RsrcData: RsrcArray;
                       var AltData: AltPointer);

{-----------------------------------------------------------------------------}
{   Creates a linked list of resource structure alternatives.                 }
{                                                                             }
{   Calling Routines:  program MAINTOP                                        }
{   Called Routines:   procedure NextRsrc                                     }
{-----------------------------------------------------------------------------}

var I,J: integer;                {local counter variables}
    SpecCnt: integer;            {counter for specialization strategy}
    TaskCover: TaskTstArray;     {indicator for covered tasks}
    TmpStructure: RsrcIntArray;  {temporary resource structure}
    CrntAlt: AltPointer;         {pointer to current alternative}

begin
    writeln('** Building Resource Structures ...');
```

```
    AltData:=nil;
    SpecCnt:=0;
    for I:=1 to NumRsrc do TmpStructure[I]:=0;
    writeln(Out,'Specialization   Resources');
    writeln(Out,'   Strategy      Employed');
    for I:=1 to NumRsrc do begin
        for J:=1 to NumTask do TaskCover[J]:=false;
        NextRsrc(I,TaskCover,SpecCnt,NumMach,NumTask,NodeLimit,NumRsrc,
          CostLimit,TaskData,NodeData,RsrcData,AltData,CrntAlt,TmpStructure)
      end; {for}
    writeln(Out);
    writeln(Out)
  end; {MakeAltSpace}

{*****************************************************************************}

procedure MakeDec(var I,NumDec: integer;
                  var TmpState: NodeIntArray;
                  var AltData: AltPointer;
                  var TmpDec: StateRealArray;
                  var Dec: DecMatrix;
                  var BasicDec: StateIntArray);

{-----------------------------------------------------------------------------}
{   Creates a nondominated feasible resource assignment decision.             }
{                                                                             }
{   Calling Routines:  procedure NextTrans                                    }
{   Called Routines:   none                                                   }
{-----------------------------------------------------------------------------}

var J: integer;           {local counter variable}
    CrntAlt: AltPointer;  {pointer to current alternative}

begin
    NumDec:=NumDec+1;
    if (BasicDec[I]=0) then BasicDec[I]:=NumDec;
    Dec[0,NumDec]:=TmpState[0];
    Dec[I,NumDec]:=0.0;
    for J:=1 to NumState do if (I<>J) then begin
        Dec[J,NumDec]:=-TmpDec[J];
        Dec[I,NumDec]:=Dec[I,NumDec]+TmpDec[J]
      end; {for}
    CrntAlt:=AltData;
    while (CrntAlt<>nil) do begin
        if not CrntAlt^.Accept then begin
            if (CrntAlt^.Excluded=nil) then begin
                new(CrntAlt^.Excluded);
                CrntAlt^.CrntEx:=CrntAlt^.Excluded
              end else begin
                new(CrntAlt^.CrntEx^.NextEx);
                CrntAlt^.CrntEx:=CrntAlt^.CrntEx^.NextEx
```

```
            end; {else}
          with CrntAlt^.CrntEx^ do begin
              ExDec:=NumDec;
              NextEx:=nil
            end; {with}
        end; {if}
      CrntAlt:=CrntAlt^.NextAlt
    end; {while}
  end; {MakeDec}

{*****************************************************************************}

procedure NextTrans(I: integer;
                    var CrntTrans: TransPointer;
                    var TmpState: NodeIntArray;
                    var AssnTask,MaxAssnTask: TaskIntArray;
                    var TmpDec: StateRealArray;
                    var NumTask,NumRsrc,NumState: integer;
                    var OpsRate: real;
                    var TaskData: TaskArray;
                    var RsrcData: RsrcArray;
                    var NodeData: NodeArray;
                    var AltData: AltPointer;
                    var Dec: DecMatrix;
                    var BasicDec: StateIntArray);

{-----------------------------------------------------------------------------}
{   Recursively considers possible transitions to build an assignment       }
{   decision.                                                               }
{                                                                           }
{   Calling Routines:  procedure MakeDecSpace, procedure NextTrans          }
{   Called Routines:   procedure MakeDec                                    }
{-----------------------------------------------------------------------------}

var J,K,L: integer;       {local counter variables}
    Accept: boolean;      {indicator for acceptable decision}
    CrntAlt: AltPointer;  {pointer to current alternative}
    UnassnRsrc: integer;  {resources not assigned to tasks}

begin
    if (CrntTrans=nil) then begin
        Accept:=false;
        CrntAlt:=AltData;
        while (CrntAlt<>nil) do begin
            CrntAlt^.Accept:=true;
            for J:=1 to NumRsrc do
              if (CrntAlt^.Structure[J]>0) then begin
                  UnassnRsrc:=CrntAlt^.Structure[J];
                  for K:=1 to NumTask do
                    if (K in RsrcData[J].TaskQual) then
                      UnassnRsrc:=UnassnRsrc-AssnTask[K]*TaskData[K].ReqRsrc;
```

```
                if (UnassnRsrc<0) then CrntAlt^.Accept:=false
                  else for K:=1 to NumTask do
                    if (K in RsrcData[J].TaskQual) and
                      (AssnTask[K]<MaxAssnTask[K]) and
                        (UnassnRsrc>=TaskData[K].ReqRsrc) then
                          CrntAlt^.Accept:=false
              end; {if}
            if CrntAlt^.Accept then Accept:=true;
            CrntAlt:=CrntAlt^.NextAlt
          end; {while}
        if Accept then
          MakeDec(I,NumDec,TmpState,AltData,TmpDec,Dec,BasicDec);
      end else begin
        J:=CrntTrans^.NextState;
        if (CrntTrans^.LossNode=0) then begin
            TmpDec[J]:=TmpState[0]*OpsRate*NodeData[CrntTrans^.GainNode].Prob;
            NextTrans(I,CrntTrans^.NextTrans,TmpState,AssnTask,MaxAssnTask,
              TmpDec,NumTask,NumRsrc,NumState,OpsRate,TaskData,RsrcData,
                NodeData,AltData,Dec,BasicDec);
          end else begin
            L:=CrntTrans^.TransTask;
            for K:=TmpState[CrntTrans^.LossNode] downto 0 do begin
                TmpDec[J]:=K*TaskData[L].TaskRate
                  *NodeData[CrntTrans^.LossNode].RedAdj;
                AssnTask[L]:=AssnTask[L]+K;
                NextTrans(I,CrntTrans^.NextTrans,TmpState,AssnTask,MaxAssnTask,
                  TmpDec,NumTask,NumRsrc,NumState,OpsRate,TaskData,RsrcData,
                    NodeData,AltData,Dec,BasicDec);
                AssnTask[L]:=AssnTask[L]-K
              end; {for}
          end; {else}
      end; {else}
  end; {NextTrans}

{*****************************************************************************}

procedure MakeDecSpace(var NumTask,NumRsrc,NodeLimit,NumState,NumDec: integer;
                       var OpsRate: real;
                       var TaskData: TaskArray;
                       var NodeData: NodeArray;
                       var RsrcData: RsrcArray;
                       var StateData: StateArray;
                       var AltData: AltPointer;
                       var Dec: DecMatrix;
                       var BasicDec: StateIntArray);

{-----------------------------------------------------------------------------}
{   Determines nondominated feasible decisions for each system state and      }
{   resource structure.                                                       }
{                                                                             }
{   Calling Routines:  procedure Optimize                                     }
```

```
{   Called Routines:   procedure NextTrans                                    }
{-------------------------------------------------------------------------------}

var I,J: integer;                {local counter variables}
    TmpState: NodeIntArray;      {temporary state data}
    TmpDec: StateRealArray;      {temporary rate data}
    AssnTask: TaskIntArray;      {number of tasks assigned resources}
    MaxAssnTask: TaskIntArray;   {maximum number of assignable tasks}
    CrntOccupy: OccupyPointer;   {pointer to current occupied node}
    CrntTrans: TransPointer;     {pointer to current transition}

begin
    for I:=1 to NumState do begin
        BasicDec[I]:=0;
        for J:=0 to NodeLimit do TmpState[J]:=0;
        CrntOccupy:=StateData[I];
        while (CrntOccupy<>nil) do begin
            TmpState[CrntOccupy^.Node]:=CrntOccupy^.Machs;
            CrntOccupy:=CrntOccupy^.NextOccupy
          end; {while}
        for J:=1 to NumTask do begin
            AssnTask[J]:=0;
            MaxAssnTask[J]:=0;
          end; {for}
        CrntTrans:=TransData[I];
        while (CrntTrans<>nil) do begin
            J:=CrntTrans^.TransTask;
            if (J>0) then
              MaxAssnTask[J]:=MaxAssnTask[J]+TmpState[CrntTrans^.LossNode];
            CrntTrans:=CrntTrans^.NextTrans
          end; {while}
        for J:=1 to NumState do TmpDec[J]:=0.0;
        CrntTrans:=TransData[I];
        NextTrans(I,CrntTrans,TmpState,AssnTask,MaxAssnTask,TmpDec,NumTask,
          NumRsrc,NumState,OpsRate,TaskData,RsrcData,NodeData,AltData,Dec,
            BasicDec)
      end; {for}
    for I:=0 to NumState do begin
        Dec[I,0]:=0.0;
        Dec[I,NumDec+1]:=0.0
      end; {for}
    for J:=0 to NumDec do Dec[NumState,J]:=1.0;
    BasicDec[NumState+1]:=NumDec+1;
    writeln(Out,'Number of Decision Variables:  ',NumDec:4)
  end; {MakeDecSpace}

{*******************************************************************************}

procedure OrderAlts(var NumState: integer;
                    var AltData: AltPointer;
                    var Dec: DecMatrix;
```

```
                    var BasicDec: StateIntArray);

{-------------------------------------------------------------------------------}
{    Employs a selection sort to order resource structure alternatives by      }
{    estimated performance.                                                    }
{                                                                              }
{    Calling Routines:  procedure Optimize                                     }
{    Called Routines:   none                                                   }
{-------------------------------------------------------------------------------}

var I,J: integer;                                {local counter variables}
    CrntAlt,MinAlt,LastAlt,TmpAlt: AltPointer;  {pointers to local alternatives}

begin
    CrntAlt:=AltData;
    MinAlt:=CrntAlt;
    while (CrntAlt<>nil) do begin
        for J:=1 to NumDec do Dec[NumState+1,J]:=0.0;
        with CrntAlt^ do begin
            CrntEx:=Excluded;
            while (CrntEx<>nil) do begin
                Dec[NumState+1,CrntEx^.ExDec]:=1.0;
                CrntEx:=CrntEx^.NextEx
              end; {while}
            GoodGuess:=0.0;
            for I:=1 to NumState do begin
                J:=BasicDec[I];
                while (Dec[NumState+1,J]=1.0) do J:=J+1;
                GoodGuess:=GoodGuess+1.0/Dec[I,J]
              end; {for}
            if (GoodGuess>MinAlt^.GoodGuess) then MinAlt:=CrntAlt;
          end; {with}
        if (CrntAlt^.NextAlt=nil) then LastAlt:=CrntAlt;
        CrntAlt:=CrntAlt^.NextAlt
      end; {while}
    while (MinAlt<>LastAlt) do begin
        if (AltData=MinAlt) then begin
            AltData:=MinAlt^.NextAlt;
            AltData^.PrevAlt:=nil
          end else begin
            MinAlt^.PrevAlt^.NextAlt:=MinAlt^.NextAlt;
            MinAlt^.NextAlt^.PrevAlt:=MinAlt^.PrevAlt
          end; {else}
        TmpAlt:=LastAlt^.NextAlt;
        LastAlt^.NextAlt:=MinAlt;
        MinAlt^.NextAlt:=TmpAlt;
        if (TmpAlt<>nil) then TmpAlt^.PrevAlt:=MinAlt;
        MinAlt^.PrevAlt:=LastAlt;
        CrntAlt:=AltData;
        MinAlt:=CrntAlt;
        while (CrntAlt<>LastAlt^.NextAlt) do begin
```

```
            if (CrntAlt^.GoodGuess>MinAlt^.GoodGuess) then MinAlt:=CrntAlt;
            CrntAlt:=CrntAlt^.NextAlt
          end; {while}
      end; {while}
  end; {OrderAlts}

{*****************************************************************************}

procedure DualSimplex(var NumState,NumDec,Spec: integer;
                      var OpMach: real;
                      var Method: char;
                      var OptAlt: AltPointer;
                      var SpecOpt: OptArray;
                      Dec: DecMatrix;
                      BasicDec: StateIntArray);

{---------------------------------------------------------------------------}
{   Determines optimal performance for a resource structure by performing   }
{   dual pivoting operations.                                               }
{                                                                           }
{   Calling Routines:  procedure Optimize                                   }
{   Called Routines:   none                                                 }
{---------------------------------------------------------------------------}

var I,J,Row,Col,NumItn: integer;  {local counter/identifier variables}
    Rate,Ratio,Min,Max: real;     {local linear programming variables}
    ShortStop: boolean;           {indicator for early exit}

begin
    for I:=1 to NumState do begin
        Col:=BasicDec[I];
        if (Dec[NumState+1,Col]<>0.0) then
            for J:=0 to NumDec+1 do
              Dec[NumState+1,J]:=Dec[NumState+1,J]-Dec[I,J]
      end; {for}
    NumItn:=0;
    ShortStop:=false;
    repeat
        Row:=0;
        Max:=-0.00001;
        for I:=1 to NumState+1 do
          if (Dec[I,0]<Max) then begin
              Row:=I;
              Max:=Dec[I,0]
            end; {if}
        if (Row<>0) then begin
            NumItn:=NumItn+1;
            Col:=0;
            Min:=99999.9;
            for J:=1 to NumDec+1 do
              if (Dec[Row,J]<0.0) then begin
```

```
                Ratio:=Dec[0,J]/Dec[Row,J];
                If (Ratio<Min) then begin
                    Col:=J;
                    Min:=Ratio
                  end; {if}
              end; {if}
            Rate:=Dec[Row,Col];
            for J:=0 to NumDec+1 do Dec[Row,J]:=Dec[Row,J]/Rate;
            for I:=0 to NumState+1 do
              if (I<>Row) and (Dec[I,Col]<>0.0) then begin
                  Rate:=Dec[I,Col];
                  for J:=0 to NumDec+1 do Dec[I,J]:=Dec[I,J]-Dec[Row,J]*Rate
                end; {for}
            BasicDec[Row]:=Col
          end; {if}
        OpMach:=-Dec[0,0];
        case Method of
            'N','n': if (SpecOpt[Spec]<>nil) then
                       if (OpMach<=SpecOpt[Spec]^.Optimum) then ShortStop:=true;
            'S','s': if (OptAlt<>nil) then
                       if (OpMach<=OptAlt^.Optimum) then ShortStop:=true
          end; {case}
      until (Row=0) or ShortStop;
    write(Out,'Expected Number of Operating Machines: ',OpMach:8:4);
    if (Row>0) then writeln(Out,' (or less)') else writeln(Out);
    writeln(Out,'Iterations: ',NumItn:4);
    writeln(Out)
  end; {DualSimplex}

{*****************************************************************************}

procedure Optimize(var NumTask,NumRsrc,NodeLimit,NumState,NumDec: integer;
                   var OpsRate: real;
                   var Method: char;
                   var TaskData: TaskArray;
                   var NodeData: NodeArray;
                   var RsrcData: RsrcArray;
                   var StateData: StateArray;
                   var TransData: TransArray;
                   var AltData: AltPointer;
                   var Dec: DecMatrix;
                   var BasicDec: StateIntArray);

{-----------------------------------------------------------------------------}
{   Determines optimal resource structure using sequential linear programming  }
{   algorithm.                                                                 }
{                                                                              }
{   Calling Routines:  program MAINTOP                                         }
{   Called Routines:   procedure MakeDecSpace                                  }
{-----------------------------------------------------------------------------}
```

```
var I,J,Row,Col,Spec,NumItn: integer;  {local counter/identifier variables}
    Rate,Ratio,Max,Min: real;          {local linear programming variables}
    TotalCost: real;                   {total cost of resource structure}
    OpMach: real;                      {number of operating machines}
    Basis: StateIntArray;              {array of basic variable identifiers}
    SpecOpt: OptArray;                 {specialty strategy optimums}
    CrntAlt: AltPointer;               {pointer to current alternative}
    OptAlt: AltPointer;                {pointer to optimal alternative}

begin
    writeln('** Optimizing ...');
    writeln(Out,'**** OPTIMIZATION SUMMARY ****');
    writeln(Out);
    NumDec:=0;
    writeln(' * Building Decision Space ...');
    MakeDecSpace(NumTask,NumRsrc,NodeLimit,NumState,NumDec,OpsRate,TaskData,
      NodeData,RsrcData,StateData,AltData,Dec,BasicDec);
    if not (Method in ['I','i']) then OrderAlts(NumState,AltData,Dec,BasicDec);
    writeln(' * Solving Initial Linear Program ...');
    for Row:=1 to NumState do begin
        Col:=BasicDec[Row];
        Rate:=Dec[Row,Col];
        for J:=0 to NumDec do Dec[Row,J]:=Dec[Row,J]/Rate;
        for I:=0 to NumState do if (I<>Row) and (Dec[I,Col]<>0.0) then begin
            Rate:=Dec[I,Col];
            for J:=0 to NumDec do Dec[I,J]:=Dec[I,J]-Dec[Row,J]*Rate
          end; {for}
      end; {for}
    NumItn:=NumState;
    repeat
        Col:=0;
        Max:=0.00001;
        for J:=1 to NumDec do
          if (Dec[0,J]>Max) then begin
              Col:=J;
              Max:=Dec[0,J]
            end; {if}
        if (Col<>0) then begin
            NumItn:=NumItn+1;
            Row:=0;
            Min:=99999.9;
            for I:=1 to NumState do
              if (Dec[I,Col]>0.0) then begin
                  Ratio:=Dec[I,0]/Dec[I,Col];
                  if (Ratio<Min) then begin
                      Row:=I;
                      Min:=Ratio
                    end; {if}
                end; {if}
            Rate:=Dec[Row,Col];
            for J:=0 to NumDec do Dec[Row,J]:=Dec[Row,J]/Rate;
```

```
        for I:=0 to NumState+1 do
          if (I<>Row) and (Dec[I,Col]<>0.0) then begin
              Rate:=Dec[I,Col];
              for J:=0 to NumDec do Dec[I,J]:=Dec[I,J]-Dec[Row,J]*Rate
            end; {for}
        BasicDec[Row]:=Col
      end; {if}
  until (Col=0);
writeln(Out,'Initial Iterations:  ',NumItn:4);
writeln(Out);
writeln(' * Starting Sequential Analysis ...');
writeln;
for I:=1 to MaxSpec do SpecOpt[I]:=nil;
CrntAlt:=AltData;
OptAlt:=nil;
while (CrntAlt<>nil) do begin
    TotalCost:=0.0;
    write(Out,'Resource Structure:  (');
    for I:=1 to NumRsrc do begin
        write(Out,CrntAlt^.Structure[I]:4);
        TotalCost:=TotalCost+CrntAlt^.Structure[I]*RsrcData[I].Cost;
      end; {for}
    writeln(Out,' )');
    writeln(Out,'Total Cost: ',TotalCost:8:2);
    for J:=0 to NumDec do Dec[NumState+1,J]:=0.0;
    Dec[NumState+1,NumDec+1]:=1.0;
    with CrntAlt^ do begin
        CrntEx:=Excluded;
        while (CrntEx<>nil) do begin
            Dec[NumState+1,CrntEx^.ExDec]:=1.0;
            CrntEx:=CrntEx^.NextEx
          end; {while}
      end; {with}
    Spec:=CrntAlt^.SpecCode;
    DualSimplex(NumState,NumDec,Spec,OpMach,Method,OptAlt,SpecOpt,Dec,
      BasicDec);
    CrntAlt^.Optimum:=OpMach;
    if (OptAlt=nil) then OptAlt:=CrntAlt
      else if (OpMach>OptAlt^.Optimum) then OptAlt:=CrntAlt;
    if not (Method in ['S','s']) then begin
        if (SpecOpt[Spec]=nil) then SpecOpt[Spec]:=CrntAlt else
          if (OpMach>SpecOpt[Spec]^.Optimum) then SpecOpt[Spec]:=CrntAlt;
      end; {if}
    writeln('Current Optimum: ',OptAlt^.Optimum:8:4);
    CrntAlt:=CrntAlt^.NextAlt
  end; {while}
writeln(Out);
writeln(Out,'OPTIMAL SPECIALIZATION STRATEGY: ',OptAlt^.SpecCode:4);
write(Out,'Resource Structure:  (');
for I:=1 to NumRsrc do write(Out,OptAlt^.Structure[I]:4);
writeln(Out,' )');
```

```
    writeln(Out,'Expected Number of Operating Machines: ',OptAlt^.Optimum:8:4);
    if not (Method in ['S','s']) then begin
        writeln(Out);
        writeln(Out);
        writeln(Out,'OPTIMAL PERFORMANCE FOR EACH SPECIALIZATION STRATEGY');
        writeln(Out);
        I:=1;
        while (SpecOpt[I]<>nil) do begin
            with SpecOpt[I]^ do begin
                writeln(Out,'Specialization Strategy: ',SpecCode:4);
                write(Out,'Resource Structure:  (');
                for J:=1 to NumRsrc do write(Out,Structure[J]:4);
                writeln(Out,' )');
                writeln(Out,'Expected Number of Operating Machines: ',
                  Optimum:8:4);
                writeln(Out)
              end; {with}
            I:=I+1
          end; {while}
      end; {if}
  end; {Optimize}

{*****************************************************************************}

begin
    open(Data,'MAINTOP.DAT',old);  {VAX/VMS Pascal}
    open(Out,'MAINTOP.OUT',new);   {VAX/VMS Pascal}
    {assign(Data,'MAINTOP.DAT');}  {Turbo-Pascal}
    {assign(Out,'MAINTOP.OUT');}   {Turbo-Pascal}
    reset(Data);
    rewrite(Out);
    writeln('RUNNING PROGRAM MAINTOP');
    ReadData(NumMach,NumTask,NumRsrc,OpsRate,CostLimit,Method,
      TaskData,RsrcData);
    close(Data);
    MakeNetwork(NumTask,NumNode,OpsRate,TaskData,LowNode,NodeData);
    NodeLimit:=NodeCap(NumMach,NumNode);
    if (NodeLimit<NumNode) then
      ReduceNetwork(NumTask,NumNode,NodeLimit,TaskData,LowNode,NodeData);
    ListNetwork(NumTask,NumNode,NodeLimit,NodeData);
    MakeStateSpace(NumMach,NodeLimit,NumState,StateData);
    MakeTransSpace(NumTask,NumState,NodeLimit,NodeData,StateData,TransData);
    MakeAltSpace(NumMach,NumTask,NodeLimit,NumRsrc,CostLimit,TaskData,NodeData,
      RsrcData,AltData);
    Optimize(NumTask,NumRsrc,NodeLimit,NumState,NumDec,OpsRate,Method,TaskData,
      NodeData,RsrcData,StateData,TransData,AltData,Dec,BasicDec);
    close(Out);
    writeln;
    writeln('ALL DONE (Output written to file "MAINTOP.OUT")')
  end. {MAINTOP}
```

B.3 Sample Output

```
**** INPUT DATA SUMMARY ****

Number of Machines:        2
Operations Rate:      0.5000
Expenditure Limit:    100.00
Solution Method:      NORMAL


Task  1
Repair Rate:          1.0000
Resources per Task:        1
Failure Rate:             --
Prerequisite Tasks:    2   3

Task  2
Repair Rate:          0.2500
Resources per Task:        1
Failure Rate:         0.2000
Prerequisite Tasks:

Task  3
Repair Rate:          0.5000
Resources per Task:        2
Failure Rate:         0.2500
Prerequisite Tasks:


Resource  1
Unit Cost:      10.00
Qualified Tasks:    1

Resource  2
Unit Cost:      20.00
Qualified Tasks:    2

Resource  3
Unit Cost:      25.00
Qualified Tasks:    3

Resource  4
Unit Cost:      30.00
Qualified Tasks:    2   3

Resource  5
Unit Cost:      33.00
Qualified Tasks:    1   2   3
```

```
**** MODEL SUMMARY ****

Number of maintenance nodes in full network:    4

Node    1
Routing Probability:   0.5263
Pending Tasks:     1
Eligible Tasks:    1

Node    2
Routing Probability:   0.1404
Pending Tasks:     1   2
Eligible Tasks:    2

Node    3
Routing Probability:   0.1880
Pending Tasks:     1   3
Eligible Tasks:    3

Node    4
Routing Probability:   0.1454
Pending Tasks:     1   2   3
Eligible Tasks:    2   3

Number of system states:     15

Specialization   Resources
   Strategy      Employed
       1         { 1 2 3 }
       2         { 1 4 }
       3         { 5 }
```

```
**** OPTIMIZATION SUMMARY ****

Number of Decision Variables:    35
Initial Iteraticns:    15

Resource Structure:  (   0   0   0   0   3 )
Total Cost:    99.00
Expected Number of Operating Machines:   0.8409
Iterations:    3

Resource Structure:  (   1   2   2   0   0 )
Total Cost:   100.00
Expected Number of Operating Machines:   0.8159
Iterations:    1

Resource Structure:  (   1   0   0   3   0 )
Total Cost:   100.00
Expected Number of Operating Machines:   0.8103
Iterations:    3

Resource Structure:  (   2   1   2   0   0 )
Total Cost:    90.00
Expected Number of Operating Machines:   0.8080 (or less)
Iterations:    3

Resource Structure:  (   2   0   0   2   0 )
Total Cost:    80.00
Expected Number of Operating Machines:   0.8027 (or less)
Iterations:    1


OPTIMAL SPECIALIZATION STRATEGY:    3
Resource Structure:  (   0   0   0   0   3 )
Expected Number of Operating Machines:   0.8409


OPTIMAL PERFORMANCE FOR EACH SPECIALIZATION STRATEGY

Specialization Strategy:    1
Resource Structure:  (   1   2   2   0   0 )
Expected Number of Operating Machines:   0.8159

Specialization Strategy:    2
Resource Structure:  (   1   0   0   3   0 )
Expected Number of Operating Machines:   0.8103

Specialization Strategy:    3
Resource Structure:  (   0   0   0   0   3 )
Expected Number of Operating Machines:   0.8409
```

References

[1] Barton, H. R. et al. *A Queuing Model for Determining System Manning and Related Support Requirements.* Technical Report AMRL-TDR-64-21. Aerospace Medical Research Laboratories, Wright-Patterson AFB, OH, Jan 1964.

[2] Bazaraa, M. S. and J. J. Jarvis. *Linear Programming and Network Flows.* 2nd ed. New York: John Wiley and Sons, Inc., 1990.

[3] Bellman, R. E. and S. E. Dreyfus. *Applied Dynamic Programming.* Princeton, NJ: Princeton University Press, 1962.

[4] Berman, M. B. and C. Batten. *Increasing Future Fighter Weapon System Performance by Integrating Basing, Support, and Air Vehicle Requirements.* Project Report N-1985-1-AF. The RAND Corporation, Santa Monica, CA, Apr 1983.

[5] Bruell, S. C. and Gianfranco Balbo. *Computational Algorithms for Closed Queueing Networks.* New York: North Holland, Inc., 1980.

[6] Brooke, Anthony et al. *GAMS: A User's Guide.* San Francisco, CA: The Scientific Press, 1988.

[7] Burke, P. J. "The Output of a Queuing System," *Operations Research, 4*: 699–709 (1956).

[8] Buzen, J. P. "Computational Algorithms for Closed Queueing Networks with Exponential Servers," *Communications of the A.C.M., 16*: 527–531 (1973).

[9] Chandy, K. M. et al. "Parametric Analysis of Queuing Network Models," *I.B.M. Journal of Research and Development, 19*: 36–42 (1975).

[10] Chandy, K. M. et al. "Approximate Analysis of General Queuing Networks," *I.B.M. Journal of Research and Development, 19*: 43–49 (1975).

[11] Courtois, P. J. "Decomposability, Instabilities, and Saturation in Multiprogramming Systems," *Communications of the A.C.M., 18*: 371–376 (1975).

[12] Denning, P. J. and J. P. Buzen. "The Operational Analysis of Queueing Network Models," *Computing Surveys, 10*: 225–261 (1978).

[13] Feller, William. *An Introduction to Probability Theory and Its Applications, Volume 1.* 2nd ed. New York: John Wiley and Sons, Inc., 1957.

[14] Gordon, W. J. and G. J. Newell. "Closed Queuing Systems with Exponential Servers," *Operations Research, 15*: 254–265 (1967).

[15] Gotz, G. A. and R. E. Stanton. *Modeling the Contribution of Maintenance Manpower to Readiness and Sustainability.* Project Report R-3200-FMP. The RAND Corporation, Santa Monica, CA, Jan 1986.

[16] Heuston, M. C. *Concepts for Estimating Air Force Manpower Requirements for Planning Purposes.* Project Number RM-2611. The Rand Corporation, Santa Monica, CA, 1960.

[17] Hillestad, R. J. *DYNA-METRIC: Dynamic Multi-Echelon Technique for Recoverable Item Control.* Project Report R-2785-AF. The Rand Corporation, Santa Monica, CA, Jul 1982.

[18] Howard, R. A. *Dynamic Programming and Markov Processes.* Cambridge, MA: The M.I.T. Press, 1960.

[19] Jackson, J. R. "Networks of Waiting Lines." *Operations Research, 5*: 518–522 (1957).

[20] Jackson, R. R. P. "Queueing Systems with Phase Type Service." *Operational Research Quarterly, 5*: 109–120 (1954).

[21] Kobayashi, H. "Applications of the Diffusion Approximation to Queueing Networks: Part 1, Equilibrium Queue Distributions," *Journal of the A.C.M., 21*: 316–328 (1974).

[22] Kobayashi, H. "Applications of the Diffusion Approximation to Queueing Networks: Part 2, Transient Queue Distributions," *Journal of the A.C.M., 21*: 459–468 (1974).

[23] Koenigsberg, Ernest. "Twenty-five Years of Cyclic Queues and Closed Queue Networks: A Review," *Journal of the Operational Research Society, 33*: 605–619 (1982).

[24] Lamb, Theodore et al. *Small Unit Maintenance Specialties for the F-16: Task Identification, Data Base Development, and Exploratory Cluster Analysis.* Technical Paper AFHRL-TP-87-23. Air Force Human Resources Laboratory, Brooks AFB, TX, Dec 1987.

[25] Law, A. M. and W. D. Kelton. *Simulation Modeling and Analysis.* New York: McGraw-Hill Book Company, 1982.

[26] Lemke, C. E. "The Dual Method for Solving the Linear Programming Problem," *Naval Research Logistics Quarterly, 1*: 36–47 (1954).

[27] Manne, A. S. "Linear Programming and Sequential Decisions." *Management Science, 6*: 259–267 (1960).

[28] Moore, S. et al. *Aircraft Maintenance Task Allocation Alternatives: Exploratory Analysis.* Technical Paper AFHRL-TP-87-10. Air Force Human Resources Laboratory, Brooks AFB, TX, Nov 1987.

[29] Muntz, R. R. "Queueing Networks: A Critique of the State of the Art and Directions for Future Research," *Computing Surveys, 3*: 353–359 (1978).

[30] Nolte, L. H., Jr. *Survey of Air Force Logistics Capability Assessment Concepts-Definitions-Techniques.* Project Report AFLMC-781029-1. Air Force Logistics Management Center, Gunter AFB, AL, Aug 1980.

[31] Phelan, J. A. *Maintenance Manpower Assessed by Stochastic Models.* M.S. Thesis. Naval Postgraduate School, Monterey, CA, Sep 1975.

[32] Purvis, R. E. et al. *Validation of Queuing Techniques for Determining System Manning and Related Support Requirements.* Technical Report AMRL-TR-65-32. Aerospace Medical Research Laboratories, Wright-Patterson AFB, OH, Mar 1965.

[33] Ross, S. M. *Introduction to Probability Models.* 4th ed. Orlando, FL: Academic Press, Inc., 1989.

[34] Sherbrooke, C. L. "METRIC: A Multi-Echelon Technique for Recoverable Item Control." *Operations Research, 16*: 122–141 (1968).

[35] Shipman, C. H. *An Investigation of the Potential Manpower Savings of Combining Air Force Specialties in Aircraft Maintenance.* Project Report ACSC-85-2390. Air Command and Staff College, Maxwell AFB, AL, Apr 1985.

[36] Taylor, J. and R. R. P. Jackson. "An Application of the Birth and Death Process to the Provision of Spare Machines." *Operational Research Quarterly, 5*: 95–108 (1954).

[37] Walrand, Jean. *An Introduction to Queueing Networks.* Englewood Cliffs, NJ: Prentice-Hall, Inc., 1988.

[38] Wilson, M. G. et al. *Optimizing Aircraft Task/Specialty Allocations.* Technical Paper AFHRL-87-46. Air Force Human Resources Laboratory, Brooks AFB, TX, May 1988.